Charles River Analytics

Final Report No. R96451
Contract No. N68936-97-C-0002

Framework for Automatic Target Recognition Optimization

Harald Ruda, Magnús Snorrason, & David Shue

Charles River Analytics
55 Wheeler Street
Cambridge, MA 02138

31 October 1997

Prepared for:
Carey Schwartz
U.S. Navy Naval Air Warfare Center — Weapons Division
1 Administration Circle
China Lake, CA 93555-6100

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	25 November 1997	Final Report, Period of Performance: 6 November 1996 - 31 October 1997

4. TITLE AND SUBTITLE

Framework for Automatic Target Recognition Optimization

5. FUNDING NUMBERS

C N68936-97-C-0002

6. AUTHOR(S)
Harald Ruda, Magnús Snorrason, & David Shue

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Charles River Analytics
55 Wheeler St., Cambridge, MA 02138

8. PERFORMING ORGANIZATION REPORT NUMBER

R96451

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Navy Naval Air Warfare Center - Weapons Division
1 Administration Circle
China Lake, CA 93555-6100

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release, distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

We have designed a framework for the optimization of Automatic Target Recognition (ATR) algorithms. Successful ATR algorithms are complex, with non-linear components and feedback between components, and thus do not lend themselves to traditional analytical optimization methods. A prototype of the designed framework has been implemented with a visual programming interface that simultaneously aids design decisions and provides opportunities for improvements and optimizations. This framework is applicable to individual algorithms, groups of algorithms, and whole ATR suites. The framework can accommodate larger systems where the ATR algorithm is but one part; it is also possible to embed the framework into a larger system. We established concept feasibility in Phase I, which specified a design and implemented a prototype for the ATR optimization framework entirely in Java. The Phase I effort included a built-in ATR taxonomy to aid algorithm design and successfully demonstrated algorithm optimization.

14. SUBJECT TERMS
Automatic Target Recognition
Optimization
Design Framework
Visual Programming Interface
Extensible Architecture
Java

15. NUMBER OF PAGES
73

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18

Abstract

We have designed a framework for the optimization of Automatic Target Recognition (ATR) algorithms. Successful ATR algorithms are complex, with non-linear components and feedback between components, and thus do not lend themselves to traditional analytical optimization methods. A prototype of the designed framework has been implemented with a visual programming interface that simultaneously aids design decisions and provides opportunities for improvements and optimizations. This framework is applicable to individual algorithms, groups of algorithms, and whole ATR suites. The framework can accommodate larger systems where the ATR algorithm is but one part; it is also possible to embed the framework into a larger system.

We established concept feasibility in Phase I, which specified a design and implemented a prototype for the ATR optimization framework entirely in Java. The Phase I effort included a built-in ATR taxonomy to aid algorithm design and successfully demonstrated algorithm optimization.

In Phase II we will enhance the ATR optimization framework in several ways: (1) Improve the basic operation of the framework to support feedback loops, a parameter editor, and the handling of image sets, (2) Extend the taxonomy and provide a taxonomy template editor to give the ATR designer freedom to construct new taxonomies, (3) Implement several choices for the optimization function, including Powell's method and Genetic Algorithms, (4) Demonstrate how to use existing C/C++ code and executables within the framework, and (5) Optimize two full ATR suites to validate the framework and to demonstrate code reuse. The result of the Phase II effort will be a fully implemented and tested ATR optimization framework.

Table of Contents

Table of Figures

Table of Tables

1 Introduction

Automatic target recognition (ATR) is the processing of information from one or more sensors in order to detect and recognize targets (Sherman et al., 1993; Bhanu & Jones, 1993 Bhanu, 1986). There have been numerous ATR applications based on various methods, including statistical pattern recognition, synthetic discriminant functions, model-based vision, neural-network-based classification, knowledge-based reasoning, and syntactic pattern recognition. In practice, these efforts have not been totally successful mainly due to non-repeatability of target signatures, competing clutter objects having similar features as valid targets, natural variability in outdoor scenarios (different terrain, vegetation, weather, etc.), target camouflage and obscuration, and limited use of a priori information (Jones, 1993; Roth, 1990). Traditional approaches—dating back to the early 1960s—of signal processing, pattern recognition, knowledge based reasoning, and neural networks have not been able to provide a robust solution (i.e. maximal detection and minimal false alarms) when target signatures and backgrounds are allowed to vary in an unknown manner.

Consequently, ATR research is an ongoing process in the Navy as well as other branches of the DoD and at many companies and universities. Given limited research budgets, new methods and algorithms often compete for funding opportunities. There thus exists a need to develop new methods and test environments to support rapid performance evaluation of developed algorithms, as well as to support design optimization and enhancement.

1.1 Objectives

The objectives of the work are to develop a framework that allows an ATR algorithm designer to optimize and compare arbitrary ATR algorithms. This Framework will:

- Enable tuning and comparison of algorithms
- Provide a graphical interface for easy manipulation of algorithms
- Provide a flexible and easily extensible (plug-in) architecture
- Provide a built-in hierarchy of relevant design choices for the specification of air-to-surface ATR algorithms
- Encourage code re-use and sharing among algorithms

1.2 Results

We have designed a framework for the optimization of ATR algorithms. A prototype of the designed framework has been implemented with a visual programming interface that simultaneously aids design decisions and provides opportunities for improvements and optimizations. This framework is applicable to individual algorithms, groups of algorithms, and whole ATR suites. The framework will be able to accommodate larger systems where the ATR algorithm is but one part; it

will also be possible to embed the framework into a larger system.

We established concept feasibility in Phase I, which specified a design and implemented a prototype for the ATR optimization framework entirely in Java. The Phase I effort included a built-in ATR taxonomy to aid algorithm design and successfully demonstrated algorithm optimization. The Phase I effort demonstrated feasibility of the concept with the optimization of a simple segmentation algorithm. An image was hand-segmented to label regions containing targets. An exhaustive search was used to find the optimal parameter values. For each set of values the segmentation was performed and the result compared to the hand-segmented target image. At the end the parameters were set to the optimal setting.

The optimization was accurate and easy to use. Furthermore, using the Framework the optimization was performed in a methodical fashion that would not be feasible if the process were controlled manually.

1.3 Outline

This report will first discuss the design of the Framework in section 2, and then the actual implementation in section 3. The taxonomy of ATR algorithms that forms the foundation for the built-in hierarchy of ATR design objects is discussed in section 4. Section 5 presents the sample optimization problem by which we demonstrated the feasibility of our approach. Finally, section 6 presents the conclusions of our work, along with recommendations for Phase II development and Phase III commercialization potential.

Section 2 describes the design of the framework. In section 2.1 we discuss the requirements of the design, which are chosen to satisfy the objectives from the previous section. In section 2.2 a sketch for a possible user interface is presented. Finally, in section 2.3 the finer points of the design are discussed.

Section 3 first reviews the benefits of object oriented programming and the benefits for this project. Section 3.1 discusses the Java language and environment and its benefits to this project. Section 3.2 describes the use of abstract interfaces in the Framework. The basic classes used in the Framework are discussed in section 3.3, and the overall class structure is presented in section 3.4. The visual interface is reviewed in section 3.5. Section 3.6 recapitulates the various ways in which the Framework is flexible and extensible.

Section 4 presents the built-in taxonomy of ATR algorithms. Section 4.1 discusses preprocessing. Section 4.2 presents the detection of potential targets. Feature extraction is discussed in section 4.3. Classification is discussed in section 4.4. Finally, in section 4.5, future options regarding the ATR

taxonomy are mentioned.

Section 5 discusses the demonstration of the prototype Framework. Section 5.1 presents the goals and structure of the demonstration. Section 5.2 presents the implementation of the demonstration. Section 5.3 makes explicit connections between the demonstration and the operation of the fully implemented prototype planned for Phase II.

Section 6 first recapitulates the specific achievements of the Phase I effort in section 6.1. The conclusions of the work are stated in section 6.2. Section 6.3 presents the recommendations for Phase II, and section 6.4 discusses the potential post applications.

Section 7 lists the references used in the text and is followed by an Appendix containing summaries of a variety of ATR algorithms.

2 Framework Design

This section describes the design of the framework. In section 2.1 we discuss the requirements of the design, which are chosen to satisfy the objectives from the previous section. In section 2.2 a sketch for a possible user interface is presented. Finally, in section 2.3 the finer points of the design are discussed.

2.1 Design Requirements

The Framework for ATR algorithm optimization and tuning must be implemented with a visual programming interface that simultaneously aids design decisions and provides opportunities for improvements and optimizations. This framework will be applicable to individual algorithms, groups of algorithms, and whole ATR suites. The framework must be easily extensible and also encourage code re-use and sharing between algorithms. The framework must also be able to accommodate larger systems where the ATR algorithm is but one part; and it must also be possible to embed the framework into a larger system.

The design requirements can be summarized by:

- Visual interface
- Built-in hierarchy
- Tuning and comparison
- Extensible architecture
- Encourage code re-use and sharing
- Embedding

In order to clarify these requirements, they are discussed in the following paragraphs:

The Framework must provide a **visual interface** for easy manipulation of algorithms. While there is no inherent requirement that ATR algorithms be manipulated using only visual programming tools — in fact, ATR algorithms are often written in traditional programming languages — it has been our experience that ATR algorithms are more easily manipulated using such tools. Our experience has been corroborated by others in the field, and visual programming tools such as Khoros are now available for general visual programming development.

Given the flexibility of visual programming, it is therefore necessary that the Framework provides guidance in the form of a **built-in hierarchy** of relevant choices for designing domain specific algorithms, in our Phase I effort, covers the air-to-surface ATR problem. This means that the ATR algorithm designer is guided at each step in the process; when a design choice is made, it is made from a list of allowed choices. However, it must also be possible for the expert ATR algorithm

designer to go beyond those choices when necessary, as the Framework is not intended to embody all available ATR algorithm design knowledge (an impossible task). As an example, at a certain point in the segmentation part of an ATR algorithm an edge-finding method may need to be used. In this case, the designer will be presented with a choice of edge-finding methods, and can then choose one of those and adjust its parameters.

The Framework should provide an intuitive interface for the **tuning and comparison** of ATR algorithms. This is the main purpose of the Framework, and it must be possible to compare different ATR algorithms on an equal footing. Furthermore, it must be possible to optimize any given algorithm using an appropriate optimization method. This also means that any combination of parameters can be optimized while the rest stay fixed.

The Framework must necessarily provide flexible and easily **extensible architecture** (plug-ins), to deal with variety of ATR algorithms. Again, just as the Framework cannot possibly embody all the ATR algorithm knowledge of even one expert, it must also be capable of accommodating new ideas that are not part of the original design. Because this event is likely to arise, the architecture for incorporating such extensions must be both simple and generic.

The Framework should **encourage code re-use and sharing** between algorithms. It should be designed in such a way that if new code is developed for a particular algorithm, it will immediately and publically become available to other algorithms that can use the same type of operation. As an example, if the designer creates a new edge-finding method in the course of designing one ATR algorithm, this method should be immediately be available anywhere an edge-finding method is required, even in another ATR algorithm suite.

Finally, the Framework must also support the **embedding** of more complex systems that include ATR algorithms as sub-systems, as well as the use of the Framework within a larger system. This is necessary because ATR systems do not exist in isolation, but may need to be tuned in concert with other systems that make use of ATR output or systems that gather and manipulate the information used by the ATR algorithms. An example of such an application would be the case where the images fed into the ATR algorithm must go through a limited bandwidth channel. The imagery may be compressed and passed through a possibly lossy channel; how does this affect the performance of the ATR algorithm? To answer this, the ATR algorithm has to be re-optimized for the new situation. This problem can be dealt with in two different ways: either the whole system, including the image compression and lossy transmission modeling, is included in the Framework; or only the ATR algorithm is included in the Framework, but the Framework is accessed in a simple way, say via a command-line interface with file inputs and outputs, such that it can be incorporated into an arbitrary system. The choice will likely depend on the non-ATR components, and it is important

that the Framework be flexible enough to be used in the most natural manner.

2.2 User Interface Sketch

This section presents a sketch for an interface that satisfies the requirements stated above. This sketch focuses on the visible part of the Framework and the direct manipulation of the visual objects that are part of ATR algorithms. Other interface components, such as menus, buttons, etc., are intentionally ignored in order to focus on the desirable characteristics of the algorithm components. Bear in mind that the specific demarcation of functions at various levels in the following example is somewhat arbitrary; the actual implementation follows a demarcation based on the taxonomy of ATR algorithms discussed in Section 4 below.

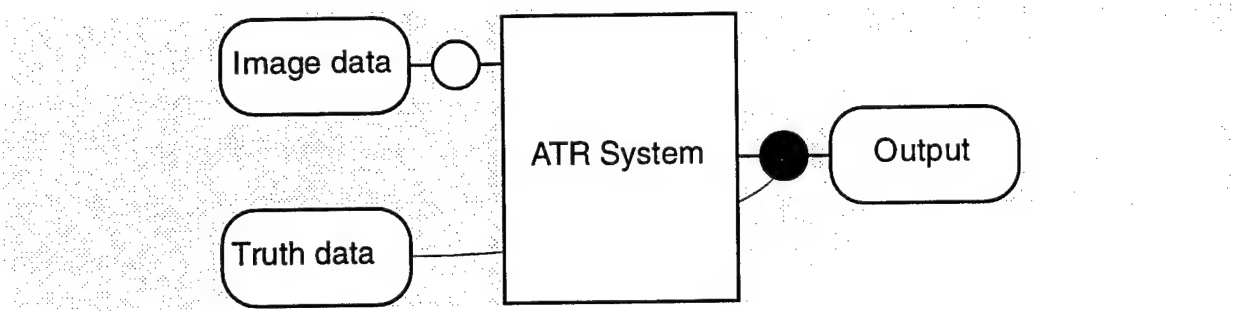

Figure 2.2-1: Top Level of the Interface

Figure 2.2-1 shows the top level with both image data and truth data as inputs needed by the ATR system. The circles show connections for the ATR Multimeter™, filled-in circles being evaluation and optimization points. The open circles are the "ground" connection that must be used for optimization; these open circles allow a part of the system to be optimized. To explain the

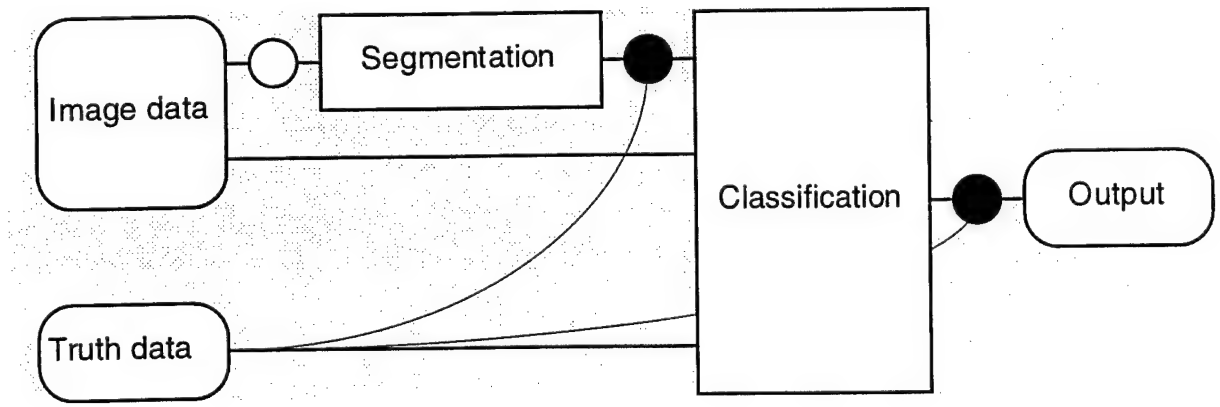

Figure 2.2-2: One Step Down

meaning of these circles in a different way, the existence a circle shows that data is available and a filled-in circle indicates that a metric is also available, so that the output can be evaluated.

Figure 2.2-2 shows the result of clicking on the "ATR System" in the previous Figure. Some of the internals of the ATR System now become visible, and more specific operations become possible. Further clicking on the "Classification" might reveal the diagram in Figure 2.2-3, where the internals of the classification sub-system becomes visible. The use of "clicking" on objects to reveal more information about them, and choosing objects to work with using visual manipulation are key elements of the proposed design.

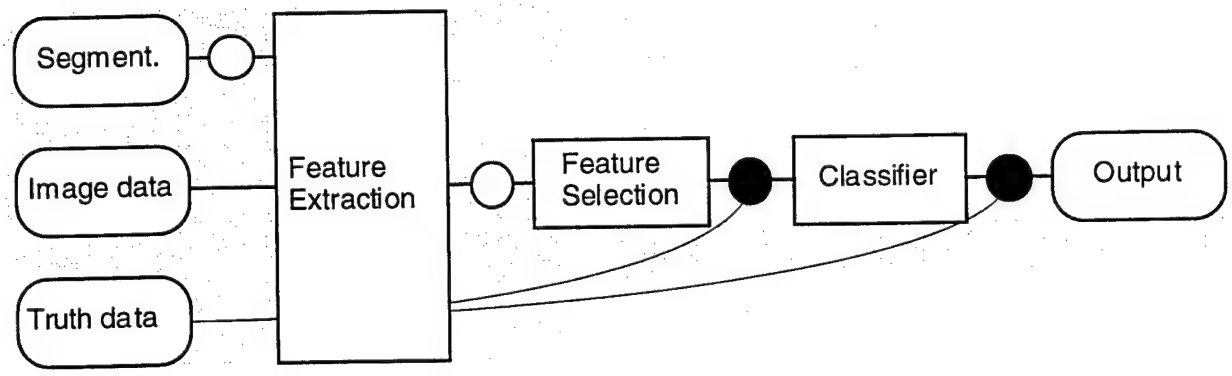

Figure 2.2-3: Stepping into Classification

2.3 Design Elements

The design requirements in section 2.1 state that a visual interface for manipulating the ATR algorithms should be used. The interface sketch just discussed builds on this idea, which is derived from working with visual dataflow environments such as Khoros and LabView. Those systems successfully use icons on a two dimensional sheet with the ability to more or less arbitrarily connect those icons. Those systems also control the execution order of the code represented by the icons so that code is executed only when valid data is available. One of the differences between Khoros and the present Framework is that the latter is designed with the hierarchical representation of algorithms as an integral feature of the system, whereas in Khoros the creation and manipulation of so-called "sub-worksheets" feels more like an afterthought, being awkward and buggy.

The part of the interface that displays the system under consideration is a panel with a number of inter-connected blocks. Each block depicts a separate sub-system, and data paths are shown as connections between blocks. Blocks have one or more data inputs on the left side of the block, and one or more data outputs on the right side of the block. It is possible to select a series of blocks that should be optimized by clicking on jacks that sit on the data paths. For each sub-system, the popup

menu can be used to select an implementation method for that sub-system. If no popup menu is available, that means that only one implementation method is available. The block is also a button, so clicking on it will open up the panel, showing the internals of this sub-system.

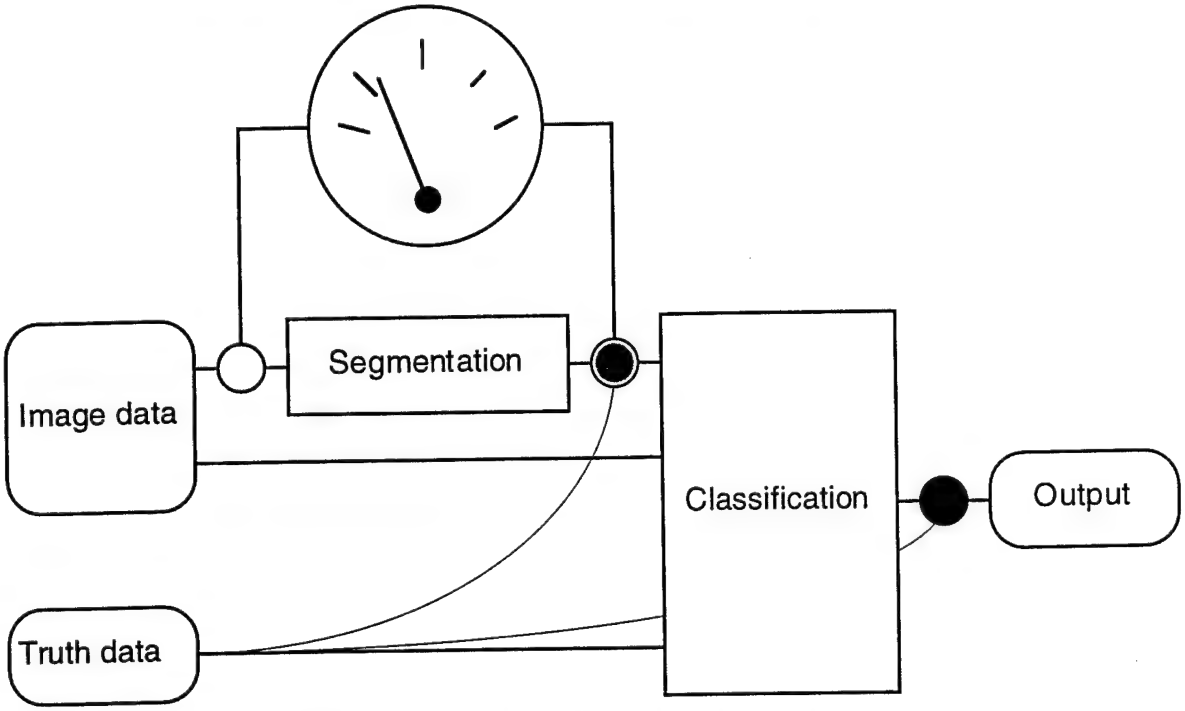

Figure 2.3-1: Hooking up the ATR Multimeter™

The visual interface used for manipulating ATR algorithms should be helpful both for guiding design decisions and making various tuning and optimization decisions. In particular, we plan to use an ATR Multimeter™ for measuring performance and evaluating the results of various optimizations. The ATR designer should be able to hook up the virtual leads of this multimeter to various points in the visual representation as shown in Figure 2.3-1, indicating where performance should be measured; optimizations are indicated in exactly the same way. Clearly, it should not be possible to measure performance at arbitrary points, as performance has to be defined and calculated. The possibility of measuring performance or performing optimization should be indicated visually, for example using a different color for the connection circle rather than the standard color (again, see Figure 2.3-1). However, functions to perform this evaluation can be added anywhere, usually with very little coding effort. These evaluation functions, whether built-in or plug-ins, are required to follow the same specification with well-defined inputs and an output in the range of 0-100%.

The Framework should provide guidance to the ATR algorithm designer, beginner and expert alike. The Framework should enforce certain design choices that can only be circumvented deliberately. For example, the Framework should only allow preprocessing type operations within a

preprocessing block. (This can be circumvented by creating a new preprocessing operation that performs the required processing.) The restrictions on design choices include restriction on the types of operations as well as restrictions on the types and number of inputs to an operation. These should help make the ATR algorithm execute correctly without significant performance deficiencies. It should also be possible for the ATR algorithm designer to override any of these restrictions.

Because the Framework cannot know all that a knowledgeable ATR designer knows, and because it is impossible to predict what types of algorithms may be developed in the future, it is important that the Framework is extensible in every possible way. Therefore it should be possible to provide new algorithms, new implementations of algorithms, new visual interfaces, even new optimization algorithms, should the need arise.

The Framework should be able to perform optimization on complete and partial ATR algorithms. The optimization should use state of the art optimization methods that have been developed in the field of numerical computing. These techniques must be multi-dimensional and include Powell's method, Genetic Algorithms, the downhill simplex method, and simple exhaustive search. Each of these algorithms has its own benefits. The main issue for ATR optimization is that each "function" evaluation is very expensive and requires the processing of each step of the ATR algorithm up to the point where it is evaluated. The exhaustive search optimization method is easy to implement, but completely inefficient. The downhill simplex method and Powell's method are nicely described in Press et al (1986). In short, the downhill simplex method uses an amoebae-like crawl through the multi-dimensional space to find a minimum. Powell's method tries to find direction of steepest descent using successive function evaluations. The method called Genetic Algorithms uses a pool of parameter value combinations (called chromosomes). For each generation a new pool of chromosomes is created based on the fitness (evaluation) of previous chromosomes, as well as possible random mutations and cross-overs. The Genetic Algorithms approach, while slow, is a good method for avoiding local minima, the constant bane of traditional optimization algorithms.

Table 2.3-1: Optimization Algorithms

Algorithm	Benefit	Drawback
Genetic Algorithms	Flexible, finds global max/min	Very slow
Powell's method	Efficient, builds up gradient info	May not find global max/min
Downhill simplex method	Simple to use	Slow, may not find global max/min
Exhaustive search	Simple to use, simple to program	Slowest of them all
Conjugate gradient	Efficient	Requires gradients, not used
quasi-Newton (DFP, BFGS)	Efficient, highly developed	Requires gradients, not used
Linear regression	Efficient for linear models	Requires gradients, not used
Marquardt-Levenberg	Efficient for nonlinear models	Requires gradients, not used
Custom method	Tuned to specific problem	Must be added

It should be possible to add new optimization algorithms to the Framework. While this is rarely going to be a desirable option, there may be cases where a specialized optimization algorithm can be constructed that is more efficient than any of the provided alternatives. Such an optimization algorithm can perhaps take advantage of private knowledge of the inner workings of an ATR suite.

In the list of optimization algorithms above, traditional methods that use gradients have not been included. The reason for this is that gradients are not readily available, except by additional expensive function evaluations. Powell's method, a non-gradient method included above, does a good job of incorporating all previous functional evaluations in order to build up this kind of information. If someday an ATR algorithm needs to be optimized which can provide gradient information, one of the gradient-based optimization algorithms (conjugate gradient, quasi-Newton, linear regression, or Marquardt-Levenberg) can be implemented as a custom optimization algorithm and used for greater efficiency, but they are not included as gradient information is not readily available.

Another special emphasis on ATR algorithm design is the Framework's ability to transparently handle sets of images. In Khoros for example, one input always corresponds to one image, and this creates a lot of extra work for the ATR algorithm designer when various loops have to be created to deal with sets of images. In the Framework, blocks will be able to automatically handle sets of images or other objects, which will also simplify the sometimes rather complex file management issues that accompany ATR algorithm design. Once the idea of image sets is introduced, the ATR designer can benefit even more by having available several different sets that can be called up by name.

In addition to handling complex sets of data the Framework should also be aware of the distinction between data used for training, evaluation, and testing. When developing a classifier that learns from examples, training data is required for the classifier to learn. A set of testing data is used to evaluate the performance of the classifier. However, if it is necessary to tune the classifier by changing some parameters until optimal performance is reached, part of the training set or a separate evaluation set must be used. If the testing set is used for this purpose, it is no longer a good predictor of the classifier performance. Knowledge about these issues will be built into the Framework, so that the ATR algorithm designer has some help in managing these distinctions.

The Framework should make full use of the object oriented features found in modern programming environments. The restrictions that are used to guide design choices can be coded into the inheritance hierarchy. This makes it easy to extend the system by creating derived classes or by implementing interfaces, an interface being a list of functions that an object can implement. The inheritance hierarchy can also be used to embody the taxonomy hierarchy, which is closely related to the design choice restrictions already mentioned. The Framework will also use a modular architecture, with separate modules for the optimization framework, GUI components, and the ATR

taxonomy. New modules can be constructed for each new ATR algorithm that is implemented.

When implementing a new ATR algorithm within the Framework, it should be clear that the implementation can be arbitrarily opaque or transparent. By this we mean that the ATR algorithm can be implemented at a high level, where none of the sub-systems are visible and manipulable; or the system can be implemented such that all sub-systems are visible with parameters and implementations that can be manipulated. A low-level, or transparent, implementation does of course involve more work, but there are benefits. With a transparent implementation, it is possible to take advantage of sub-system implementation methods that have been provided by other transparent implementations; this is not possible for opaque implementations because the sub-systems are not visible. It is also possible to control the optimization at a finer level, which can be an advantage when algorithms are large and complex. The greatest advantage is the ability to manipulate the algorithm directly to modify it in ways that merely adjusting parameters cannot achieve.

In terms of a description for programmers familiar with object oriented languages, most of the work done by the ATR algorithm designer consists of composing, creating new objects by using existing objects and putting them together. The work that goes into extending the functionality is usually done by using the inheritance mechanism.

Table 2.3-2: Design Elements and Approaches

Design Element	Approach
Algorithm editing	Visual editing of a dataflow type representation
Selection of sub-part of algorithm	Visual select beginning and end
Algorithm evaluation	ATR Multimeter™ displays the information
Algorithm design	Guided by framework, provide lists of choices
Algorithm variability	Built-in taxonomy accommodates wide range of algorithms
Unknown algorithms	Extensibility, add new objects
Algorithm optimization	Choose from optimization algorithms
File handling, train/test/evaluate data distinction	Automatic handling of sets of images
Complex taxonomy of algorithm components	Object oriented inheritance hierarchy

3 Framework Implementation

This section first reviews the benefits of object oriented programming and the benefits for this project. Section 3.1 discusses the Java language and environment and its benefits to this project. Section 3.2 describes the use of abstract interfaces in the Framework. The basic classes used in the Framework are discussed in section 3.3, and the overall class structure is presented in section 3.4. The visual interface is reviewed in section 3.5. Section 3.6 recapitulates the various ways in which the Framework is flexible and extensible.

Based on the requirements described in section 2, we determined that indeed it is preferable to use an object oriented environment. The object oriented facilities of data abstraction, inheritance, and polymorphism are all well suited for the ATR optimization framework. Data abstraction enforces modularity, so that code is split into independent modules with clearly defined methods used for communication between modules. The inheritance mechanism provides a way to inherit behaviors from other modules, reducing the amount of programming by promoting code reuse between modules. Polymorphism allows the inherited modules to have somewhat different behaviors than their parent modules. All in all, these features simplified the implementation of the Framework tremendously.

In addition, for the implementation of the Framework we also required an environment where the details of a graphical user interface (GUI) are taken care of. Preferably, there should be a standard set of calls that would work on any platform.

3.1 Java

As a result of matching the design requirements with different implementation options, we chose to use the Java programming language and environment for the Phase I effort. There are a number of good arguments for using Java, including:

- Productivity; some companies report as much as a 10-fold gain over C++! This is because Java is less error prone than C++, as it has garbage collection, memory management, and a pure object oriented architecture
- Cross-platform portability, including a cross-platform GUI
- Testability; the pure object oriented architecture enforces modularity
- Straightforward multi-threaded model
- Robustness and security
- Upgradeability on the fly using the dynamic binding; useful for plug-ins and extensions
- Easy interface with legacy code, C/C++ Java Native Interface (JNI)
- Legacy executables easily wrapped in Java objects

In particular, the memory management, object oriented nature and the cross platform GUI are

reasons why we decided to go with Java. These features are not available in C++ without non-standard libraries or other additions. Because Java is an interpreted language, there are some concerns about speed. However, Java interpreters are becoming better and better, using just-in-time compilation and other techniques to run Java programs very quickly. In addition, Java compilers that compile to machine code for specific processors are also becoming available. While such code would not be portable without recompilation, this is not a great concern for this application, as the application would probably remain on a single machine or set of machines for long periods of time. In any case, the bottlenecks will be in the code that performs the details of the ATR algorithms, such as the code in the image processing library. If this code is in C/C++ then it will already be fast, if it is in Java, then it can be compiled to native.

As already mentioned, the use of Java does not preclude the use of legacy code in the implementation of ATR algorithms within the Framework. In fact, Java is perhaps already one of the most compatible environments. In particular, the ATR Framework will be able to use existing code in the form of:

- Java code, trivially compatible
- Existing libraries, for example DLLs; accessed using the Java Native Interface (JNI)
- Existing C/C++ code; also accessed using the JNI
- Executables, these will be wrapped in Java objects and executed. Communication of inputs and outputs will be through files. It is not recommended to use any interactive executables.
- Ada and Cobol now have compilers that compile to Java byte-code.

3.2 Interface Declarations

In order to isolate the actual workings of framework modules from the various messages that they can send to each other the implementation uses Java interfaces. Java provides the ability to specify interfaces, which only declare methods without providing definitions for them, much like pure virtual functions in C++. The interfaces can then be implemented by other objects by providing definitions for those methods. Java has introspective abilities that allows the Framework to figure out at run-time whether a given object implements one or more of these interfaces.

We have used interfaces to achieve the desired functionality. The interfaces used in the Framework are:

- Optimizing — The main interface, which contains methods to deal with parameters, execute functions, interact with the Framework, and of course, lets objects be optimized.
- Evaluating — Used to mark modules that can be measured. These are also the modules that can be at the end of a chain of modules to be optimized.
- SetOperable — Implemented by modules that can operate on sets of data.

Listing 3-1 shows the Optimizing interface, which is the only interface that is required of

```
public interface Optimizing
{
    /**
     *  The part of the interface that establishes how to deal with parameters
     */

    public abstract int          getParameterCount();

    public abstract String       getParameterName(int index);
    public abstract Object       getParameterType(int index);
    public abstract Object       getParameterValue(int index);

    public abstract Object[]     getParameterRange(int index);
    public abstract Object[]     getValidParameterRange(int index);

    public abstract void         setParameterValue(int index, Object value);

    public abstract boolean      isValidParameterSet();
    public abstract int[]        getInvalidParameters();

    /**
     *  The part of the interface that is used to actually execute functions
     */

    public abstract Object[]     execute(Object[] inputs);

    public abstract int          getInputCount();
    public abstract int          getOutputCount();

    public abstract int          getRequiredInputCount();

    /**
     *  These methods need to be implemented in order to fit in the framework
     */

    public void                  setFramework(Framework theFramework);
    public boolean               showParameters();
    public void                  rebuild();

    public void                  setSystemName(String theName);
    public String                getSystemName();

    public void                  addInput(String label);
    public void                  addOutput(String label);

    /**
     *  Methods required for Optimization algortihms
     *
     */

    public double                maximizeFunction(double[] param);
    public void                  setOptimizationSetSize(int numParams);
    public void                  mapOptimizationParameter(int opt, int sys);
}
```

Listing 3-1: The Optimizing Interface

modules. As can be seen, there are four main parts to this interface. The first section declares all the methods needed to access the parameters of the module. Both the parameter editor and the optimization algorithm need to use these methods when communicating with the module. The second set of methods are used to actually execute the code of the module. These methods are used by the Framework as well as by the optimization algorithm. The third set of methods are needed to work inside the Framework, and only the Framework makes use of them. The last set of methods are used only for the actual optimization step.

Because the components that make up the Framework also have a visual representation, they must derive from the `java.awt.Panel` class. The methods in this class are used to display the component and to provide the event handling for the component. The Panel class already implements the basic functionality required, simplifying the development of new components.

Section 3.3 below describes a default implementation of a component called Optimizer that derives from Panel class and implements the Optimizing interface. This component is used frequently within the Framework. The easiest way to add a component with new functionality is to create a class that derives from Optimizer and overrides the necessary methods.

The use of interfaces makes the Framework modular and very easily extensible. For example, to provide a new module with functionality that is very different from what is available from the Optimizer class, one can create a new class that derives from Panel (or derives from a class that derives from Panel) and implements the Optimizing interface. The methods that implement the interface can be completely different from the implementation in Optimizer, and thus allows for complete flexibility. It is possible to imagine a component that does not follow the design of the rest of the Framework, but that still provides a visual interface as well as a programmatic interface to be used by the Framework.

```
public interface Evaluating
{
    /**
     *  The interface for marking components that can be evaluated
     */

    public abstract double evaluate(Object [] outputs, Object truth);
}
```

Listing 3-2: The Evaluating Interface

The interface shown in Listing 3-2 is used to mark components that can be evaluated. This means that the component has a method called "evaluate" which compares the output of the component with some truth data. The actual implementation of this method depends on the nature

of the output produced by the component. This is another way in which the Framework is extensible, by providing customizable evaluation methods. Thus it is possible to provide new components and perform optimization on those components using custom evaluation methods; these operations can therefore work on data types that were never considered by the designers of the Framework.

```
public interface SetOperable
{
    /**
     *  The part of the interface that is used to actually execute functions
     */

    public abstract Hashtable[] setExecute(Hashtable[] inputs);
}
```

Listing 3-3: The SetOperable Interface

Similarly, the SetOperable interface shown in Listing 3-3 is used to mark components which operate directly on the whole set of data. In this case, the "setExecute" method is used instead of calling the "execute" on each piece of data. This can result in simpler procedures and more efficient processing of large data sets. Again, it is an optional interface that can be implemented for reasons of efficiency and convenience.

3.3 Basic Classes

This section describes those classes which are fundamental to the implementation of the Framework. While the total number of classes that support the implementations is much greater (over 50 classes in addition to Java core classes), these classes provide the essential functionality that makes the Framework a successful demonstration of the technology as well as an implementation that will serve as a starting point for the Phase II work. The functionality of the following classes is described: Framework, Optimizer, ChoiceButton, Block, ATRLayout2, OptimizedSystem, SetOptimizer, and ExhaustiveSearch.

Framework. This is the main class of the Framework. It can be used either as an applet or an application. The Framework provides services as well as an environment for the display and manipulation of ATR algorithms. Among other things, this environment takes care of the dynamic loading of classes, and other file operations.

Optimizer. This class implements all the basic functionality that is required of a system level description. The Optimizer class derives from Panel and extends Optimizing, thus it knows how to draw its contents properly. It also contains all the methods needed to deal with parameters, how to

get/set them, find the ranges and names of them, and also to deal with parameters of sub-systems. It is a complete class that can easily be extended to provide real functionality.

ChoiceButton. This class implements a new visual interface element that is a combination of a button and a popup menu. In essence, it allows the user to choose, using the popup menu, the action that will accompany a click on the button. The class is used mostly to choose the particular algorithm to implement a certain functionality within the framework, such as choosing a particular neural network architecture to implement the classifier. Another example of where this class is used is for choosing the optimization algorithm to use. The designer uses the menu to pick the type of algorithm to use for optimization; and clicking on the button then performs the optimization using the chosen algorithm.

Block. The Block class is actually derived from the ChoiceButton class and implements the sub-system visual interface component described above. When the contents of a system are explored, the result will often be a number of simpler sub-systems hooked up as to perform a certain function. The major additions to the ChoiceButton class includes methods for dealing with the number of inputs and outputs, and where those connections should go on the graphical representation.

ATRLayout2. This class is used by the Optimizer class to draw the view of the system with its components. This LayoutManager currently handles all types of feedforward systems, with multiple parallel paths, including the joining and splitting of paths.

SetOptimizer. This class extends Optimizer class and implements SetOperable interface. It is a class which can easily be used to provide set handling functionality. One great advantage of this structure is that if a class is derived from SetOptimizer, it does not need to provide a definition for "executeSet()", as that is already done by SetOptimizer. It suffices to provide a definition for "execute()" which only takes a single n-tuple of inputs. The executeSet() method of SetOptimizer will call execute() as appropriate.

OptimizedSystem. This class extends SetOptimizer and is intended to be the class from which all top level classes are derived. Thus it is also how top level systems are recognized. The Framework will look for classes derived from OptimizedSystem and list those in the top level ChoiceButton. Note that while all top level ATR systems should derive from this class, it is also possible to have other systems that do not perform ATR but that could benefit from similar types of optimization derive from this class and be included in the top level menu.

ExhaustiveSearch. The ExhaustiveSearch class is an implementation of an algorithm to actually perform the optimization by evaluating different combinations of values for the variables that can be changed. It is an example of how the optimization algorithm makes use of the facilities provided by

the framework. Despite being inefficient, this class can be used in real ATR design situations if only a small number of parameters are to be changed, or alternatively if the number of possible values for each parameter is small. This is the one class in this list of key classes that will be supplanted in the Phase II effort. While this exhaustive search optimization algorithm will still be available, the ATR designer is more likely to use one of the more efficient and versatile algorithms that will be included in the Phase II effort.

3.4 Class Structure

There is a class structure implied in the previous section. Figure 3.4-1 below shows all classes deriving from the Optimizer class. The classes in the figure include the classes that are essential to the framework operation (class names starting with cra.atro.* in the Figure) as well as other classes that implement the built-in hierarchy that aids ATR design decisions (class names starting with cra.atr.hierarchy.* in the Figure).

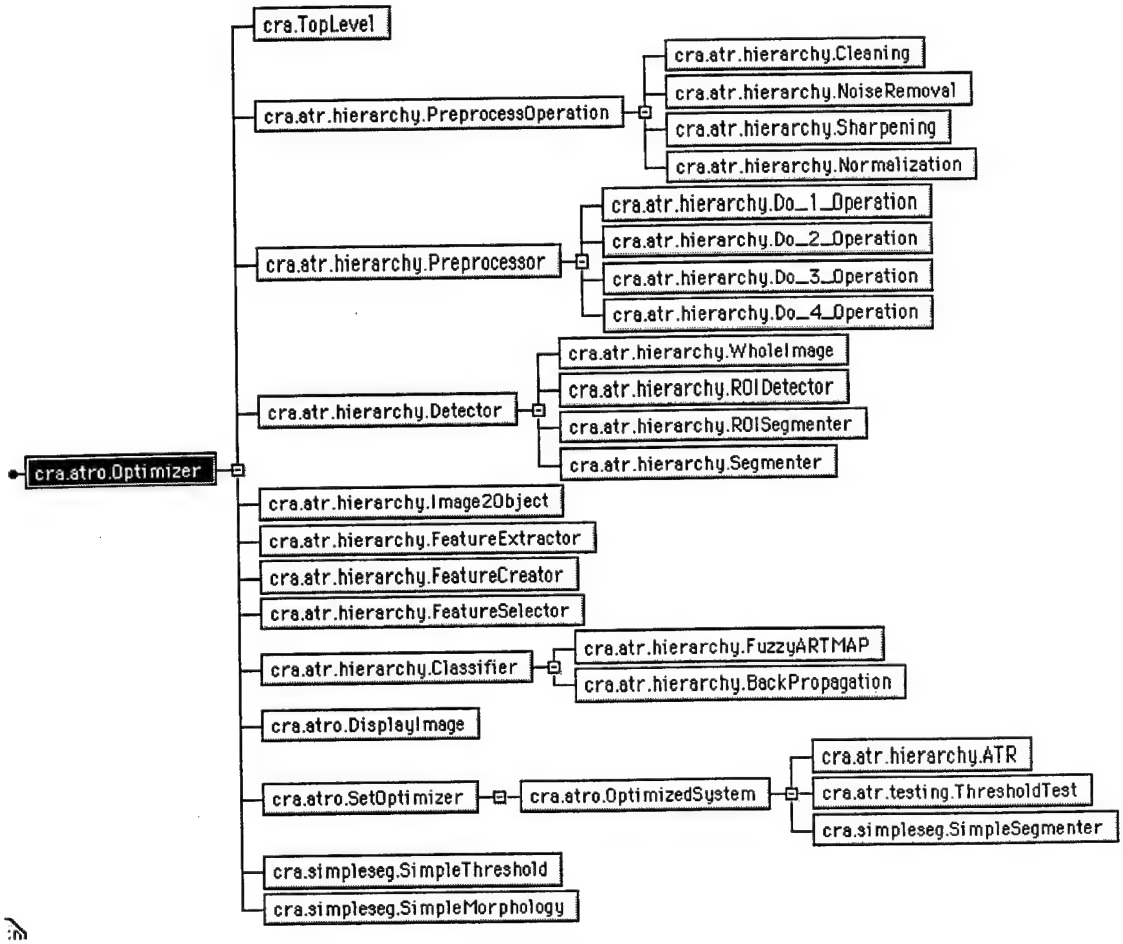

Figure 3.4-1: Class Optimizer and Derived Classes

The structure shown in Figure 3.4-1 also demonstrates the great benefit of the inheritance feature of object oriented programming. By defining one object (Optimizer) that performs the basic functions needed for a certain type of module; it is easy to create a multitude of different modules with slightly different behaviors. Each of the derived classes shown in the figure has only minimal additions to the base class (Optimizer), yet has markedly different behavior. Furthermore, it is possible to create new modules with still different behaviors without changing any of the code in the already existing modules, just as the derived classes in the figure did not require any changes to the base class.

3.5 Visual Interface

The visual interface for the design uses familiar GUI components such as buttons and popup menus. Figure 3.5-1 below shows the visual interface of the Phase I framework prototype. The components of the interface have the following functions:

Back — Return to the previous panel. This button brings the designer back up one level. It is intended to work very much like its namesake button in a web browser.

Rescan — This button should only be used when new code is added to the Framework while it is running. This button causes a scan of all the code available to the project, and makes it available in the appropriate menus. This function is run automatically when the Framework is started. This functionality is possible because of the dynamic binding used in the Java environment.

Run — This button is used to execute the system. It will execute all processes from left to right with the data being passed from each output to the next input.

Parameters — Click on this button to see all the parameters of the currently visible portion of the ATR system. When chosen at the top level, this will display all the parameters in the ATR system. The parameter editor allows the parameters to be set to any value within the allowed range.

Measure — Use this button to evaluate the system at the point selected by hooking up the ATR Multimeter™ to an appropriate red jack (red jacks indicate that an evaluation function is available). The ATR Multimeter™ will display the result of the evaluation, always on a scale from 0 to 100. A consistent scale is used so that the designer does not have to learn a new measuring system for each evaluation function.

Optimize — Use the popup menu to choose an optimization algorithm, then click on the button to perform the optimization. As different combinations of values of the parameters are tried, the ATR Multimeter™ will display the result of evaluating the current set. Currently only the exhaustive search optimization routine is available.

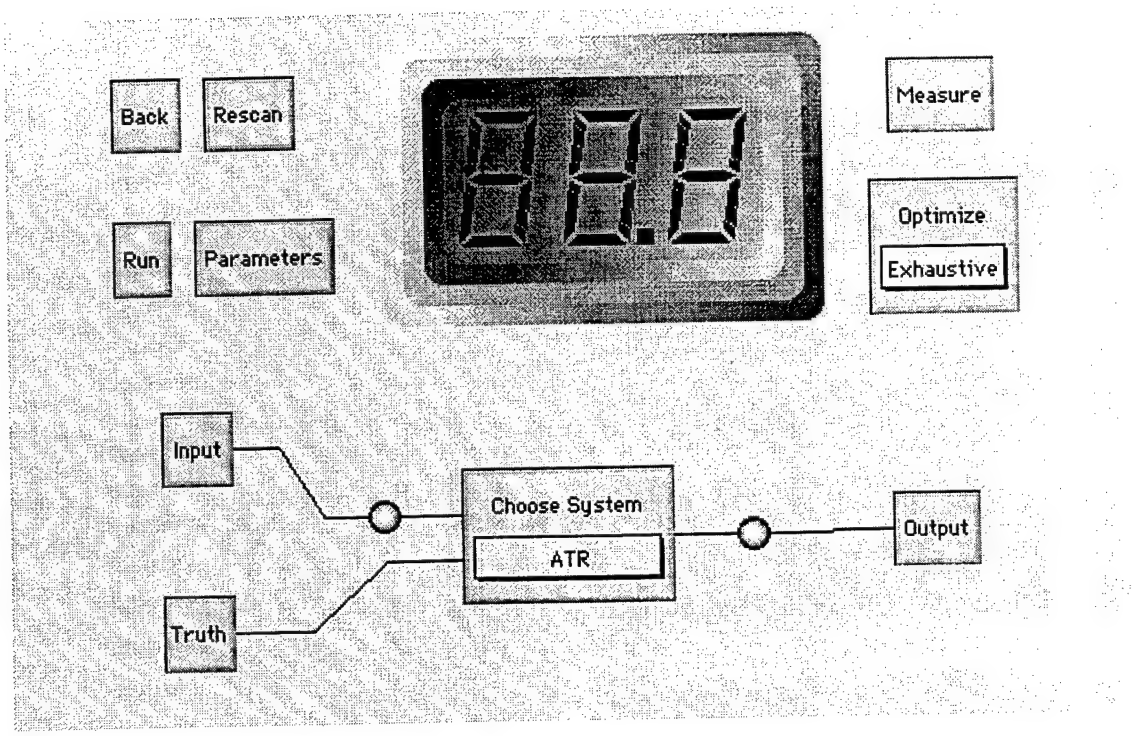

Figure 3.5-1: The Framework Interface

The designer can of course use the popup menus of sub-systems to change their implementation, and then click on the sub-system to examine and possibly modify its internal functionality. Clicking on the Input or Truth buttons brings up editors that allow the designer to select images to be part of the current image set.

3.6 Flexibility and Extensibility

As a means of summarizing the design of the Framework for ATR optimization, it is perhaps simplest to list the means in which this system is flexible and can be extended to suit the sophisticated ATR algorithm designer. The designer can choose implementations (if there is more than one option), and parameters can be adjusted to account for any particular requirements. However, true extensibility requires that the system can be used in designs that were not included in the original specification. Specifically, the ways in which this Framework can be extended are:

- Extending functionality by sub-classing existing classes and adding new code
- Adding evaluation functions where desired for optimization fine-control
- Adding new templates to the built-in taxonomy structure
- New visual interfaces that still work within the Framework
- Adding new optimization algorithms

Most of this extensibility comes from using an object oriented design approach. Furthermore, the simplicity and straightforwardness with which such extensions can be added is a tribute to the Java language and environment. The dynamic nature of the language also makes it possible to add such extensions "on the fly", that is, while the framework is already running. Thus rather than performing some lengthy saving and restoring operations that a restart of the Framework would require in some circumstances, new code can be added and incorporated quickly and efficiently.

4 ATR Taxonomy

This section presents the built-in taxonomy of ATR algorithms. Section 4.1 discusses preprocessing. Section 4.2 presents the detection of potential targets. Feature extraction is discussed in section 4.3. Classification is discussed in section 4.4. Finally, in section 4.5, future options regarding the ATR taxonomy are mentioned.

A taxonomy of ATR algorithms was constructed. The goal was to clarify the relationships between different ATR algorithms, enabling us to identify functional blocks that stay consistent across different ATR suites. The taxonomy was then hierarchically built up by applying the same process recursively to each functional block. Our approach included the following steps:

- Conducting a literature review to collect a list of algorithm suites for the task of air-to-ground ATR. The summary of a number of algorithms can be found in Appnedix A.
- Identifying the different ways used in those suites to partition the overall ATR task.
- Identifying partitions that provide interfaces between functional blocks with consistent meaning across different suites. (Such as the partition between *segmentation* and *feature extraction*, which always specifies an interface that passes isolated regions of interest (ROIs) or "objects" from segmentation to feature extraction.)
- For each partition, identifying all standard algorithms capable of providing the functionality required for each block. (I.e., for an "edge-detector" block, identifying standard approaches to edge-detection.)

We fully realize that any such taxonomy will be colored by our biases derived from greater exposure to some ATR methods than others. The literature survey was a conscious effort on our part to reduce any such biases, but it is unrealistic to expect the resulting taxonomy to be fully objective and give equal weight to all styles of ATR. One solution is to think of our taxonomy as a template that represents one or more styles of ATR. The framework can have multiple such templates, with the other templates representing ATR styles preferred by other researchers.

The end result of our taxonomy construction was a hierarchy that has four branches coming out of the root node. Figure 4-1 below shows a specific instance of our framework using one block from each of the four branches: *Preprocessing*, *Potential Target Detection*, *Feature Extraction*, and *Classification*. Top-level ATR block diagrams with the same or similar blocks in the same order are common in the literature (Rogers et al., 1995; Bhanu, 1986).

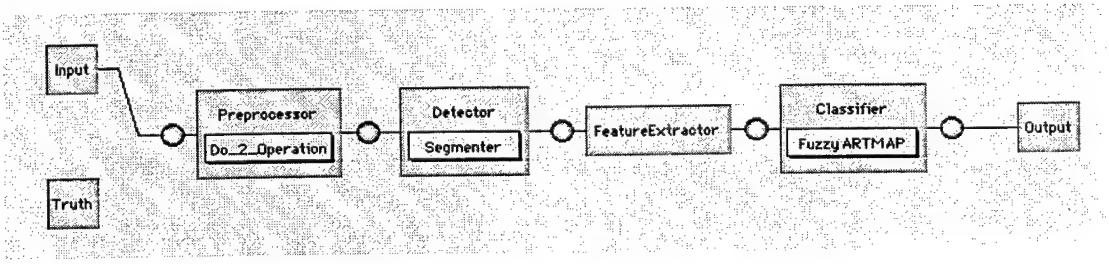

Figure 4-1: Top level of ATR algorithm taxonomy

We consider the first two blocks optional and the latter two required since we identified ATR suites without preprocessing or potential target detection, but no algorithms without some form of feature extraction or classification (according to our definition of those terms). Most real-world ATR suites, however, contain all four blocks and the following four sections of this report describe each block. The hierarchical nature of the taxonomy is represented in the outline of the following sections comprising the remainder of this chapter:

```
4.1 Preprocessing
4.2 Detecting Potential Targets
    4.2.1 Point-of-Interest (POI) Detection
        4.2.1.1 Local Contrast Analysis
        4.2.1.2 Shape Matching
    4.2.2 Segmentation
        4.2.2.1 Boundary Detection
        4.2.2.2 Region Detection
    4.2.3 POI Detection Followed By Segmentation
    4.2.4 No Operation
4.3 Feature Extraction
    4.3.1 Gray-Level Features
    4.3.2 Shape Descriptors
4.4 Classification
    4.4.1 Model Based Classification
    4.4.2 Feature Based Classification
        4.4.2.1 Supervised Classification
        4.4.2.2 Unsupervised Classification
```

4.1 Preprocessing

The first functional block of an ATR system—the *preprocessor*—gets its input directly from the sensor subsystem. In most cases this is a single image stream, but for radar sensors such as SAR or LADAR and for multi-sensor assemblies there can be multiple streams representing fundamentally different scene measurements: reflected light, emitted heat, or range. The first two are energy measurements that can be made at different frequencies and may hence constitute more than one stream each. Hence, we assume that the preprocessing block's input data consists of one or more image streams. We define an image stream as a continuous sequence of image maps where:

- pixels are arranged on a grid which tiles the sensor's whole field-of-view (FOV)
- each pixel's value represents a measurement from the corresponding local region in the FOV

The measurements in each image can be noisy, biased, saturated, obliterated by sensor artifacts, etc. The pixel grid can be nonuniform, skewed, or dynamically distorted, and crosstalk effects between pixels are common. There can also be co-registration problems between image streams, irregular sampling intervals, and dynamic changes due to mechanical vibrations & drift.

The primary goal of preprocessing is to remove those problems. Some of the more common image operations towards this goal are:

- Artifact/drop-out removal
- Noise removal
- Brightness/Contrast normalization
- Sharpening/de-blurring

A secondary goal is to *enhance* the image in a way that helps separate objects from background. In general, the image shows zero or more targets, localized clutter that may look like potential targets, background that surrounds targets and clutter, and superimposed noise. The objective of image enhancement is to increase target-to-background contrast, without making the clutter look more target-like.

Other forms of preprocessing are common for radar sensors that produce range maps and for multisensor assemblies, such as:

- Image registration
- Resampling/resolution conversion
- Coordinate conversion (such as spherical to Cartesian)
- 3-D data re-projection (such as creating a virtual top-down view)

Given the large number of possible operations and the lack of prescribed order (the exact order of preprocessing operations is not always important), we chose to implement this block as a sequence of operations without any restriction on the number or order of operations. Figure 4.1-1 below shows an example of a preprocessing block instantiation in our framework with two operations: noise removal followed by sharpening. The input and output datatypes for this particular instantiation are identical since neither operation changes image dimensions or resolution. This is not always the case, but we assume the output always consists of one or more image streams as defined above. Finally, although unusual, the possibility of ATR suites without any preprocessing exists (for example with highly advanced sensor assemblies). This option is included in our framework by specifying zero operations for the preprocessing block, in which case it simply passes the input image streams unchanged to the output.

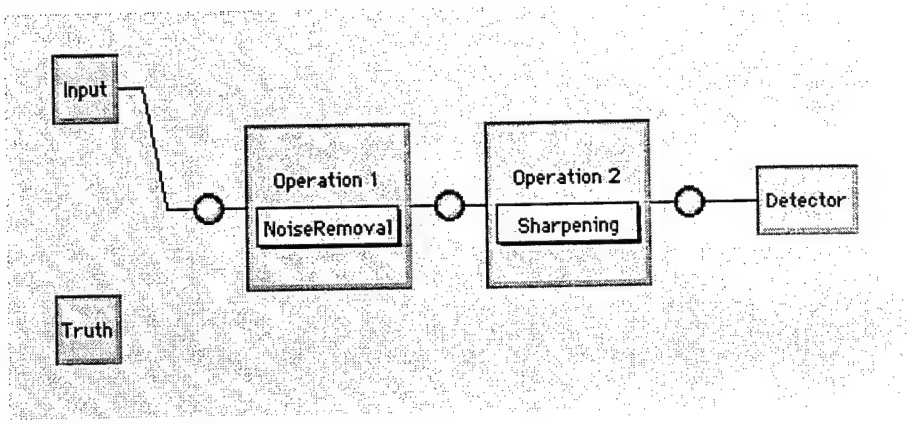

Figure 4.1-1: Preprocessing block with two operations

4.2 Detecting Potential Targets

The next top-level functional block can be identified by the need for resource allocation. The 3rd and 4th top-level functional blocks in ATR systems, the feature extractor and classifier, are often so computationally expensive that it is not feasible to apply them everywhere in the FOV. Hence, the *potential target detector* is applied first to identify the image regions most likely to contain targets: the regions-of-interest (ROI). For a large FOV sensor and relatively uncluttered scenes, the combined area of all ROI can be orders of magnitude smaller than the total FOV area.

Another reason for potential target detection is to limit the false alarm rate, which may become unacceptably high if the system looks for targets everywhere in the FOV. By limiting the number of locations at which feature extraction and classification is applied, the total number of classification decisions is also restricted and hence the upper bound on false alarms is lowered.

We identified four mutually exclusive subblocks for the potential target detection block, as shown in Figure 4.2-1 below. These subblocks represent four possible operations or combinations of operations: point-of-interest (POI) detection, segmentation, POI detection followed by segmentation, and no operation. The following four sections describe each subblock. The output of this block—regardless of the subblock choice—is a set of ROIs for each input image, where each ROI is assumed to contain one potential target.

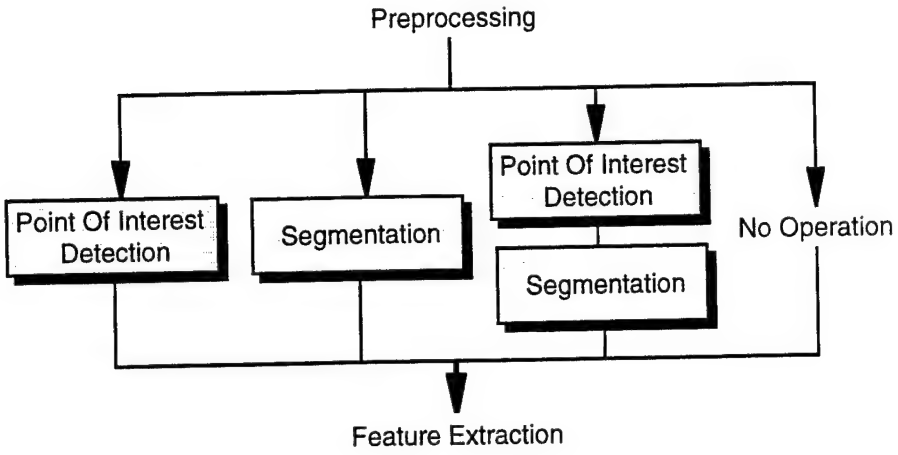

Figure 4.2-1: Subblocks inside *potential target detection* block

4.2.1 Point-of-Interest Detection

Since the main purpose of an potential target detection block is resource allocation, some ATR designers have found a source of inspiration in nature's solution to the same resource allocation problem: foveal vision. The visual systems of all higher vertebrate animals have much higher acuity in the center of the FOV—the fovea—than in the periphery, but the visual angle subtended by the fovea is only a few percent of the eye's total FOV. Thus the resource allocation issue: where in the FOV to apply this spatially restricted resource. The ultimate solution is to point the fovea at the locations where targets are most likely to be. The real problem is in estimating target probabilities for each location, i.e. to build a probability map for the FOV. Given such a map, the fovea should be centered on the highest peak.

Thus, the conceptual goal of POI detection is to compute such a target probability map and locate the highest peaks. We came across many algorithms fitting this goal in our literature review. Two-dimensional localization is most common, but given enough ranging information (such as from SAR, LADAR, or optic flow computations) 3-D localization is also possible. We identified two fundamentally different types of POI detection: those based on *local contrast analysis* and those based on *shape matching*. As indicated in Figure 4.2-2 below, we chose to present those two types of POI detection as individual subblocks and as one combination inside the POI detection block. The three options are explained in the following three sections.

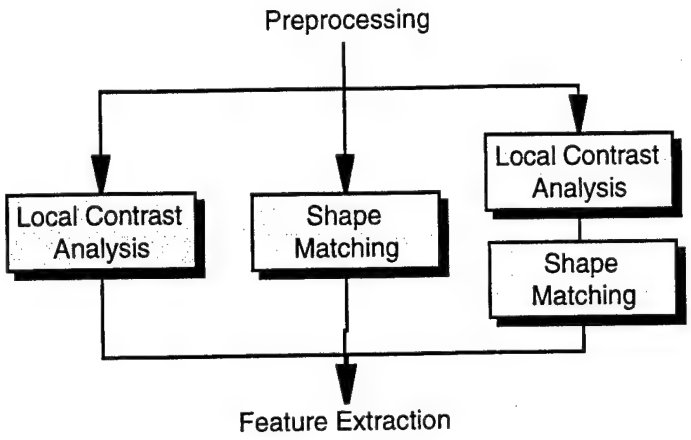

Figure 4.2-2: Subblocks inside the *point of interest detection* block

If POI detection is the only operation in the potential target detection block, then one final operation is needed to turn the POI into a ROI: a region centered on the POI needs to be identified. This can simply be a fixed size square (box for 3-D) or circle (sphere for 3-D), or its size can be scaled according to projected target size if range information is available.

4.2.1.1 Local Contrast Analysis

Local contrast can be used to eliminate regions from consideration. A large image region (larger than a target) with all pixels of the same value (after correcting for low-level random noise) cannot be an ROI; it probably belongs to the sky, smooth ground, water surface, etc. Conversely, compact regions of high contrast can be valuable target indicators. For example, in infrared imagery, the temperature difference between a hot tank and its immediate background often shows up as localized high contrast, indicating a POI (Sherman et al., 1993). Local contrast is also computed in many radar based detection systems by adaptive threshold Constant False Alarm Rate (CFAR) processors, which report a POI when a range cell's temporally-averaged signal exceeds the surrounding range cells' signal by fixed amount (Koch, Moya, Hostetler & Fogler, 1995).

4.2.1.2 Shape Matching

Approximate matching of target's overall shape can be performed many different ways. One example, the VARTAC system (Hecht-Nielsen & Zhou, 1995) describes a Gabor jet approach where a set of Gabor kernels of eight orientations and four spatial scales are applied at each grid point. These Gabor jet measurements are compared to precomputed Gabor jets for geometric shapes that roughly resemble targets. Grid points where similarity is high are designated POI's.

Target-part matching uses template-based pattern matching methods to search for target parts

that have easily identifiable shapes, such as gun barrels or treads (Sherman et al., 1993). Some specific algorithms include *normalized correlation, symmetric phase-only matched filtering,* and the *generalized Hough transform* (Ritter & Wilson, 1996). The last two come in many variations, some of which are rotation and scale invariant

4.2.1.1 Local Contrast Analysis Followed By Shape Matching

Since pattern matching algorithms are fairly computationally expensive, a more efficient approach is to use local contrast analysis first. At the very least, local contrast analysis can be used to eliminate uniform regions from the shape matching algorithm. Applying shape matching after local contrast analysis also has the potential to decrease false alarm rate. This is because the template based shape matching is more selective than the "data driven" contrast analysis.

4.2.2 Segmentation

Potential target *segmentation* attempts to maximize the difference between foreground and background, in effect generating a target outline or silhouette. The shape of the target outline is often an important feature for target recognition and the target silhouette can be used to mask out everything but the target for further processing. Of course, this same outline also defines a ROI, hence enabling the potential target detection block to contain only a segmentation operation.

Target models used for segmentation can be subtle and implicit, such as assuming connectivity of an object's edges, the existence of an *inside* and an *outside* of a two-dimensional figure, etc. The models can also be quite explicit, wherein a dictionary of target silhouettes is maintained, and optimally manipulated (i.e., scaled, translated, rotated) and selected to provide the best match to the imaged object. Thus, a variety of segmentation techniques have been brought to bear on the problem, each with their own underlying assumptions regarding the 2-D or 3-D image attributes of the object, and each with their own algorithms for target image segmentation.

A few attempts have been made to construct a taxonomy of segmentation techniques (Rosenfeld, 1984; Lindley, 1991; Kasturi & Jain, 1991). A distinction used by Kasturi & Jain, 1991 to classify techniques is to discriminate between *edge* based methods and *region* based methods. The former type is conceptually based on finding *dissimilarities* in some property of the image, while the latter type indicates a search for *similarities.* This distinction has roots in one of the fundamental questions of vision: Do you first find the boundary between regions or first distinguish the regions to localize the boundary? Using edge-based methods implicitly argues for finding the boundaries first while region based methods go by the assumption that you find the region first. The former school of thought has extensive following in machine vision and AI under the term "edge detection". The latter school of thought has following both in machine and human vision research, typically under

the term "textural grouping". As indicated in Figure 4.2-3 below, we consider boundary vs. region detection to be subblocks inside the segmentation block. They are not mutually exclusive, although most systems tend to use just one or the other.

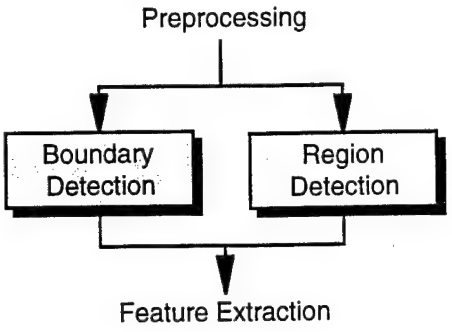

Figure 4.2-3: Subblocks inside *segmentation* block

4.2.2.1 Boundary Detection

The basic definition of the problem which any edge or boundary based method tries to solve is given by (Kasturi & Jain, 1991) as marking the pixels that lie on the boundaries of an object. The definition of a boundary in these terms is that the neighborhoods on either side of the boundary must have some dissimilar characteristics. Boundaries are therefore functions of at least two variables: neighborhood *size*, and the *metric* by which to measure the characteristics of the neighborhoods.

Boundary detection algorithms are often modular, consisting of three operations in sequence:

1. Edge detection
2. Boundary completion or gap closing
3. Labeling of closed boundaries

Edge detection algorithms use a great variety of approaches to measure local gradients of pixel values. Some are based on first partial derivatives, such as the Sobel operator (Sobel, 1970), while others are based on second partial derivatives, such as the Laplacian.

The output of such edge detection algorithms for a typical target is not a closed boundary but many edge-pieces with gaps between them. That is why a boundary completion or gap closing algorithm is also needed. Such algorithms include the Boundary Contour System (Grossberg, Mingolla & Williamson, 1995), the CORT-X 2 filter (Bradski & Grossberg, 1995), a multiscale Gabor approach (Manjunath & Chellappa, 1993), and the Grouping Field approach (Heitger & von der Heydt, 1993).

Finally, separate closed boundaries should be individually labeled ("object 1", "object 2", etc.). That is easily accomplished using a boundary follower algorithm. Note that this step may not be

needed if the overall boundary detection algorithm is only applied around one POI.

4.2.2.2 Region Detection

Kasturi & Jain, 1991 define a region as a "set of connected points that belong to the same object." Forming regions is complementary to detecting edges in the sense that it is based on a metric which compares some characteristics of different image points for *similarity*, rather then *dissimilarity* as in edge detection. Unlike edge detection, region detection needs no concept of neighborhood. Marking every pixel in the image as either belonging to a region or not is sufficient. This is not to say that the concept of neighborhood can not be incorporated into the region forming process, only that unlike edge detection techniques, it is not required.

Region detection algorithms are often modular, consisting of three operations in sequence:

1. Thresholding
2. Region growing
3. Region labeling

Thresholding is a very simple operation, but automatically choosing the pixel value at which to threshold is nontrivial. Some common methods are based on histogram analysis (threshold between peaks in the histogram) or on edge pixel analyses (threshold at the average value of pixels located on object edges) (Sherman et al., 1993).

Region growing can be based on simple morphological operations (dilation and/or erosion) to remove small holes, gaps, and very small isolated regions. It can also refer to more complex iterative algorithms that attempt to grow and merge subregions until the rate of change drops, indicating an "optimum segmentation" (Sherman et al., 1993).

As in boundary detection, a final step is needed to label separate regions. This can by accomplished using any region-filling or blob-coloring algorithm (Ballard & Brown, 1982).

4.2.3 POI Detection Followed By Segmentation

If an ideal object detection algorithm provides the coordinates of a *point* inside the object (often near the object's center), then an ideal object segmentation algorithm provides the object's *outline*.

Many segmentation algorithms are computationally expensive and can only be used efficiently when getting a hint about where to look for the potential target boundary or region. Therefore, ATR designers often pair up a POI detection algorithm with a segmentation algorithm, such that the segmentation algorithm is only applied in the spatial vicinity of a POI.

4.2.4 No Operation

The final option for this functional block is to ignore it and pass the (possibly preprocessed) input image stream directly to the next top-level functional block. In practice, this is only feasible with narrow FOV sensors that have already been cued by a separate target search system such that a large part of the FOV contains a potential target. For example, a dedicated target search system might use a wide FOV passive sensor to determine POIs and then scan each POI with a narrow FOV laser-radar sensor for a high resolution range map of the potential target. In that case it is acceptable to treat the whole FOV as a single ROI for further processing.

4.3 Feature Extraction

Feature extraction computes a vector of features, or "signatures," from the ROI specified by the potential target detector. The underlying assumption here is that if the feature vectors are statistically separable into distinct categories in the feature space, then the objects will be separable in the object space; i.e., they will be classifiable. Thus, the power of the features to provide adequate target distinguishability is critical, no matter what classification scheme is chosen. The key issue here is in the selection of features to extract, since there is a huge variety of feature types that can easily be computed as a function of the ROI. Note that this selection is sometimes linked to the choice of classification method in the next top-level functional block, some classifiers are optimized for data types that are only provided by specific feature types.

We subdivided the feature extraction block into two subblocks based on the basic feature types: *value* features and *shape* features. The subblocks are not mutually exclusive (but they require different input data), they should rather be thought of as complementary sources of signature information.

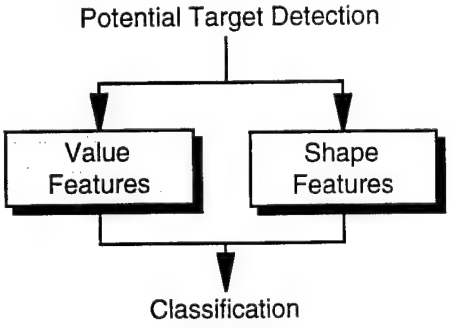

Figure 4.3-1: Subblocks inside the *feature extraction* block

The feature extraction block marks the end of image processing in the ATR processing chain. Output data from the feature extraction block consists of *vectors* or abstract *structures*, not images.

4.3.1 Value Features

For ROI images that contain more than two possible values, many features can be computed as a function of the distribution of these values. In particular the *1st order statistics* are frequently used, these are listed in Table 4.3-1 below.

Table 4.3-1: 1st Order Statistics

1st Order Statistics
Mean
Median
Variance/Standard Deviation
Min/Max
Skewness
Kurtosis
Histogram
Entropy

Much more information is contained in features that are functions of both the values and spatial relationships between ROI pixels. Such features include those listed in Table 4.3-2 below.

Table 4.3-2: Features Encoding Pixel Values and Spatial Relationships

Complex Features	Examples
2nd Order (Dipole) Statistics	Gray Level Co-occurrence Matrices
Moments	Central Normalized Standard Invariant
Transform Signatures	Gabor Wavelet Hough Karhunen-Loeve Discrete Cosine Fast Fourier Walsh Hadamard Mellin
Fractal Dimension	
Symmetry	

Detailed descriptions for many of those features can be found in Gonzalez & Woods, 1992, Ritter & Wilson, 1996, and in manuals for rapid algorithm development environments such as

Khoros and MATLAB. Symmetry as a feature of high information content is described in Reisfeld, Wolfson & Yeshurun, 1995.

4.3.2 Shape Features

For ROI images that contain just two values (pure black & white, or binary images), a number of features can be computed to describe the spatial relationships and shapes of connected regions. Note that all of those features can also be computed for grayscale ROI images if they are thresholded first, hence thresholding is common in the segmentation block to produce both grayscale and binary versions of the ROI. Table 4.3-3 shows a partial list of *region descriptor features* applicable to ATR.

Table 4.3-3: Region Descriptor Features for ATR

Region Descriptors	Attributes
Area	
Compactness	
Principal Axis	
Skeleton (Medial Axis Transform)	
Best Fitting Ellipse/Rectangle	Size Length-to-Width Ratio Orientation
Topology	Euler Number Adjacency Relations Inclusion Relations
Quadtree	

Another way to look at the ROI output from a segmentation block is to consider the only the *shape of the ROI boundary* and not the content. Table 4.3-4 shows a partial list of *boundary descriptor features* applicable to ATR.

Table 4.3-4: Boundary Descriptor Features for ATR

Boundary Descriptors	Attributes
Perimeter	Length Curvature Fractal Dimension
Fourier Descriptors	
Chain Code	
Shape Number	

Descriptions for implementing those features can be found in Ritter & Wilson, 1996 and Gonzalez & Woods, 1992.

4.4 Classification

Target *classification* is the function of the final top-level block. The goal is to label the detected potential targets based on their associated set of feature parameters. Different ATR tasks have different levels of discrimination and hence different sets of labels to choose from. At a minimum, potential targets must be classified into "target" or "clutter" categories, but generally a higher level of discrimination is expected. Typical discrimination requirements are "tank" vs. "truck," or "T-72 tank" vs. "BMP-2 tank".

A variety of classification schemes have been implemented and evaluated in past ATR efforts, including classical linear and quadratic classifiers, cluster analyses, synthetic discriminant functions, and knowledge-based discriminators, to name a few. Each provides distinct advantages in different situations, but each is beset with particular problems because of underlying implicit assumptions in their scope of applicability. This variety has led to many taxonomies and comparisons for classification methods in general (Duda & Hart, 1973; Bishop, 1995; Schürmann, 1996), and for ATR in particular (Sherman et al., 1993; ATRWG, 1991; Bhanu, 1986).

We chose to construct the simple taxonomy of classification methods shown in Figure 4.4-1 below: *model* based vs. *feature* based methods. This choice is not based solely on intrinsic differences between the methods. After all, model based methods use features too and feature based methods must contain some form of target representation. Our choice is based on a difference in *emphasis* and on a long-standing "*cultural*" difference in the field of target recognition. The former refers to the emphasis on detailed off-line target modeling and iterative on-line performance in model based methods, as opposed to the emphasis on optimized off-line training and one-step on-line performance in feature based methods. The latter refers to the top-down methodology preferred by designers of model based systems vs. the bottom-up approach preferred by designers of feature based systems.

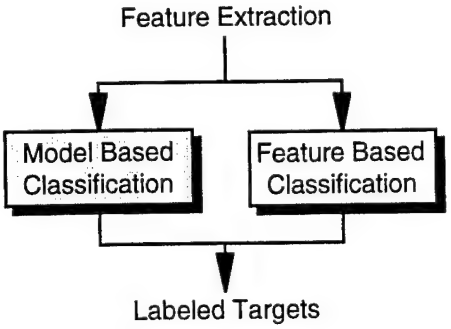

Figure 4.4-1: Subblocks inside the *classification* block

4.4.1 Model Based Classification

The model based approach to ATR classification is characterized by a reasoning process involving hypothesis generation and testing. Figure 4.4-2 below shows a sample diagram of the subblocks inside the model based block. All the blocks but for *model based reasoning* are required for a typical model based classifier. That one optional block constitutes an "outer loop" which iterates over the hypothesis generation-and-test cycle, but with higher levels of model fidelity on each iteration.

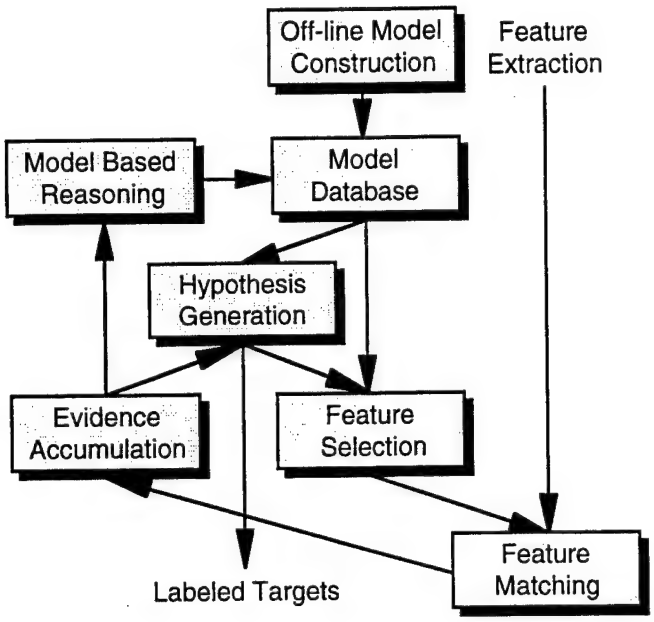

Figure 4.4-2: Subblocks inside the *model based classification* block

The key issue for the *off-line model construction* and *model database* blocks is the type of representation that the ATR designer uses. There are six main categories, shown in Table 4.4-1.

Table 4.4-1: ATR Model Representation Types

Representation
Statistical
Syntactic
Relational
Geometry
Physics
Behavioral

The task of *hypothesis generation* algorithms is to predict what features should be found a given ROI image, based on the accumulated evidence about target identity and pose. Once a hypothesis

has been generated, the appropriate set of features is pulled from the model database by the *feature selection* block. For ATR systems that recognize many targets in any pose, the feature space can be huge. Feature selection must therefore use very efficient indexing techniques, such as geometric hashing.

The selected feature can then be matched against the features actually extracted from the ROI image in the previous top-level block. This matching is usually a straightforward root-mean-square error or correlation measure per feature, performed by the *feature matching* block. That measure is forwarded to the *evidence accumulation* block, which uses one of the methods listed in Table 4.4-2 to handle uncertain data.

Table 4.4-2: Methods for Handling Uncertain Data

Method
Kalman filter
Bayes net
Dempster-Schafer
Fuzzy logic
Rule based

Closing the loop, the accumulated evidence is forwarded to the hypothesis generation block, where it either confirms or rejects the current hypothesis (Sherman et al., 1993; ATRWG, 1991).

4.4.2 Feature Based Classification

The feature based approach to ATR classification compares a precompiled multi-dimensional decision space of all known targets to the feature vector that was extracted from the ROI image. Figure 4.4-3 below shows the two primary options: the use of *supervised* or *unsupervised* classifiers. Supervised classifiers are trained off-line with sample images of targets (in multiple poses) where each target is identified by its correct label. The purpose of the training is to establish a mapping between regions in the input feature space and target labels. Unsupervised classifiers are also trained off-line with sample images of targets, but rather then enforcing a mapping, the classifier identifies inherently robust clusters in the input feature space. A postprocessing block, *category interpretation*, then analyses each cluster to identify the appropriate label (target type or clutter).

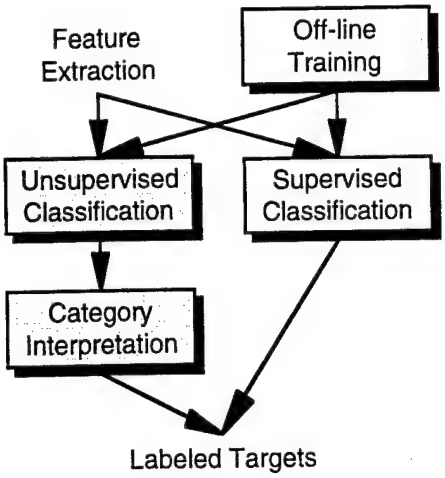

Figure 4.4-3: Subblocks inside the *feature based classification* block

The set of tasks needed for *off-line training* depends the data source as shown in Table 4.4-3.

Table 4.4-3: Off-line Data Sources and Training Steps

Data Source	Steps
Acquire database	1. Partition into training/evaluation/testing data
Synthesize from target/clutter models	1. Generate data representing backgrounds 2. Generate target/clutter data representing different types/poses 3. Combine target/clutter data with background data 4. Record truth data 5. Partition into training/evaluation/testing data
Capture Using sensor	1. Choose locations 2. Capture data with targets in multiple poses 3. Record truth data 4. Partition into training/evaluation/testing data

4.4.2.1 Supervised Classification

There is a huge variety of established methods for supervised classification. Two large groups are statistical and neural network classifiers, but there is a number of other methods. Table 4.4-4 lists a number of well-known examples of these methods.

Table 4.4-4: Supervised Classification Methods

Group	Methods
Statistical	Expectation maximization Maximum likelihood Bayes optimal
Neural network	Radial Basis Functions ARTMAP Backpropagation Boltzmann machine
Other	K-nearest neighbors Case based reasoning Decision trees Genetic programming

4.4.2.2 Unsupervised Classification

There is also great variety of unsupervised classification methods, also known as clustering algorithms, shown in Table 4.4-5.

Table 4.4-5: Unsupervised Classification Methods

Group	Methods
Optimizing prototypes	K-means Isodata ART Learning vector quantization
Neighbor distance	Nearest neighbors Furthest neighbors Mutual neighbors Density estimate Graph theory

The postprocessing algorithms used by the category interpretation block to label each cluster are typically either based on rule based inferencing or a learned mapping.

4.5 Future Options

A potentially useful task would be identifying the "perfect performance" mode for each block. For example, thermal segmentation should isolate scene regions fitting a specific thermal signature, while range segmentation should isolate scene regions that stick out of the ground. A model for a block can be constructed if its perfect performance mode is known. This is valuable information for optimization of a multi-block chain: the model can replace the block during optimization to measure how much the other blocks in the chain contribute to the overall error.

5 Demonstration of Prototype Operation

This section discusses the demonstration of the prototype Framework. Section 5.1 presents the goals and structure of the demonstration. Section 5.2 presents the implementation of the demonstration. Section 5.3 makes explicit connections between the demonstration and the operation of the fully implemented prototype planned for Phase II.

In order to demonstrate that the limited Framework prototype developed during the Phase I effort can actually perform optimization and tuning of ATR algorithms, we have developed a test demonstration. A simple ATR segmentation algorithm is used. An evaluation method that compares the result of segmentation with truth data is also provided, with the truth data in this case being a hand-segmentation of the input data.

5.1 Sample Demonstration of Optimization

The various elements that need to be specified for ATR optimization are:

- ATR algorithm
- Parameters to optimize
- Evaluation function
- Search algorithm
- Criteria for optimization

As stated above, the **ATR algorithm** used in this demonstration is exceedingly simple. It performs segmentation by a global binary threshold. Values in the image that are above the threshold go to zero (background), and values that are below the threshold value are set to one (foreground).

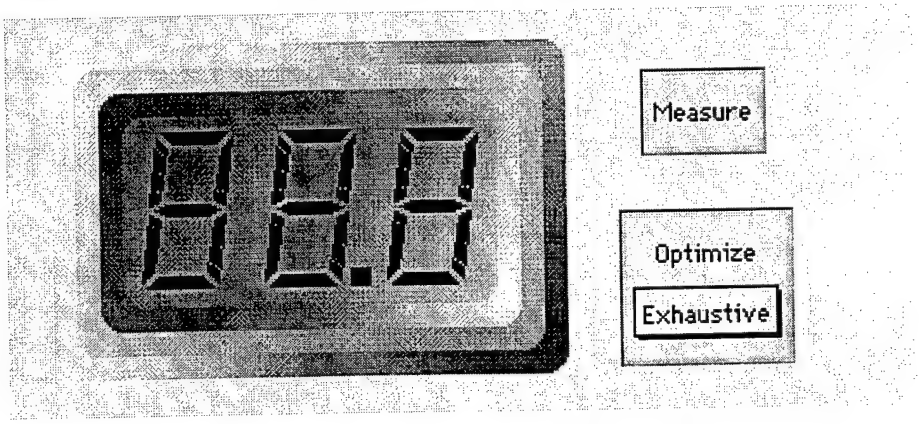

Figure 5.1-1: The ATR Multimeter™

The only **parameter** to optimize in the above algorithm is the threshold value. Optimizing only one parameter keeps the demonstration simple and manageable.

The **evaluation function** is used to compare the output of the segmentation to the truth data. In this sample demonstration, the output of the evaluation function (and shown on the ATR Multimeter™ as in Figure 5.1-1) is proportional to the number of pixels that are identical in the "truth" image and the segmented image. This is a rather simplistic evaluation of segmentation performance, and more sophisticated evaluation functions will be provided in the Phase II effort.

The **search algorithm** used was a exhaustive search. In order to cover the parameter space effectively, the search algorithm tries a fixed number (in this case 100) values between the allowed minimum and maximum values for each parameter. The exhaustive search actually evaluates every combination of allowable parameter values for those parameters that are to be optimized. Since exhaustive search optimization is very inefficient, we deliberately chose a problem with only one parameter. Optimizing k parameters using this method is $O(n^k)$ in time, which leads to drastically longer search times as more parameters are optimized.

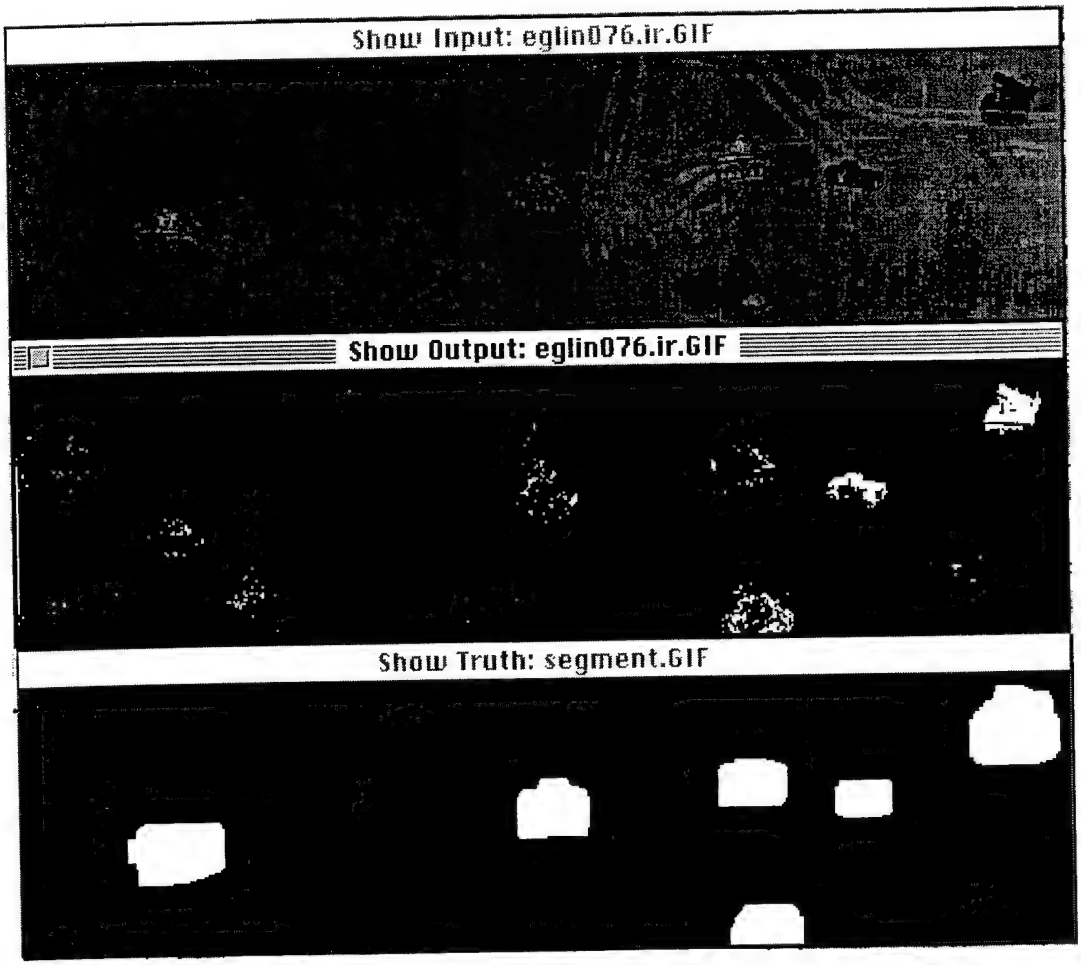

Figure 5.1-2: Input, Output, and Truth Images

The criteria used for the optimization in the current demonstration is the hand-segmented version of the input data. The hand-segmentation is rough, but accuracy is not required in order to demonstrate that the optimization works.

Figure 5.1-2 above shows the input and output images of this optimization. The input image is shown at the top; it is the infrared reflectance image of a laser radar sensor. The type of sensor used is irrelevant to the optimization Framework, but may be very relevant to parts of the ATR algorithm used. A rough hand-segmented mask is shown at the bottom. This image is used by the evaluation as a "truth" image. The middle image shown in the figure is the optimal output of the simple segmentation, as generated by the Framework. The image demonstrates the importance of optimization. Even this simple segmentation algorithm can be coaxed to produce an image with some coverage of each target and very little false alarm coverage. Bear in mind that this thresholding output can easily be cleaned up with morphological operations.

5.2 Sample Implementation

The implementation of the demonstration shown above is straightforward. The code required to implement the thresholding segmentation is shown in Listing 5.2-1, and the code required to display the thresholding as a sub-system and to perform the evaluation of the segmentation output is shown in Listing 5.2-2. As can be seen, the code is fairly simple and is really only glue code that interfaces between the Framework and the image processing routines.

```
public class Threshold extends Optimizer
{
    public void definition()
    {
        addParameterDouble("threshold", 100, 0, 255);
    }

    public Object[] execute(Object[] input)
    {
        Object[] output = new Object[1];

        int threshold = ((Number) getParameterValue(0)).intValue();

        output[0] = cra.ip.Histogram.threshold((cra.ip.Image) input[0],
                                               threshold, 255, 0, 255);

        return output;
    }
}
```

Listing 5.2-1: Code for the Threshold Component

The DoSegment class shown in Listing 5.2-2 can derive from (extend) the ROIDetect class

directly. For this demonstration, it is included in another class which derives from ROIDetect; just as the Threshold class is included in the DoSegment class in the listing.

```
public class DoSegment extends Optimizer implements Evaluating
{
    public void definition()
    {
        addSubSystem("Threshold", "Threshold");
    }

    public double evaluate(Object [] outputs, Object truth)
    {
        Image difference = Histogram.subtract((cra.ip.Image) outputs[0],
                                              (cra.ip.Image) truth);
        int count = Histogram.nonzero(difference);
        int size = difference.getWidth() * difference.getHeight();

        return 1.0 - count / (double) size;
    }
}
```

Listing 5.2-2: Code Demonstrating Use of Threshold and Evaluation Function

The image processing routines that are used in this demonstration were developed for this demonstration, the operations that are performed are not very complex. For the Phase II effort, the plan is to use a commercially available image processing library. All the same, we anticipate that the glue code used will look substantially similar to the code shown in the listings.

5.3 Comparison with Planned Phase II Operation

It is interesting to compare the elements of this demonstration with the planned features of the Phase II functional prototype:

The **ATR algorithm** will be chosen and customized by the user with the help of the visual programming interface. The customization will be completely flexible, yet it will be guided by the Framework's built-in taxonomy of ATR operations. The expert user will be able to override any built-in restrictions after being warned of any inconsistencies that may be introduced.

The **parameters** to optimize will be chosen by using the GUI to select the parameters that can and/or should be optimized. This can be just one parameter, it can be all parameters that are part of the ATR system, or it can be any arbitrary set of those parameters.

The user will probably choose to optimize the whole ATR system, in which case the **evaluation function** that will be used will be the one provided with the Framework to evaluate the output of an ATR system. More specifically, it will evaluate the resulting confusion matrix.

The **search algorithm** used will also be chosen by the user, and will most likely **not** be the exhaustive search. It is more likely that the user will choose a more efficient algorithm such as Powell's method, genetic algorithms, or the downhill simplex method.

The **criteria** to use for optimization will again be truth data that must be supplied. The data will correspond to the stage at which the algorithm is evaluated for optimization, most likely this will be at the end-stage, where targeting information output by the algorithm will be compared to the previously determined desired targeting output.

6 Conclusions & Recommendations

This section first recapitulates the specific achievements of the Phase I effort in section 6.1. The conclusions of the work are stated in section 6.2. Section 6.3 presents the recommendations for Phase II, and section 6.4 discusses the potential post applications.

6.1 Summary of Phase I Effort

The Phase I effort completed the following tasks:

- A thorough literature review of a variety of ATR algorithms
- Construction of a taxonomy of ATR algorithms
- Specification of the requirements for a framework for ATR optimization
- A design for a Framework for ATR optimization
- A prototype implementation of the ATR optimization Framework
- Development of a code base to be used in a Phase II effort
- A demonstration of the prototype implementation

In particular, the prototype implementation dealt with some of the more difficult issues, such as: how to create an easily extensible framework; how to structure a taxonomy of ATR algorithms; image set handling; dynamic loading of new code; how to use arbitrary optimization routines. Given that these problems were tackled in the Phase I effort, the Phase II effort can build directly on the existing Framework and extend it. In particular, new optimization algorithms can easily be added by following the example algorithm used in the demonstration.

6.2 Phase I Conclusions

We feel that the Phase I effort was successful in achieving its stated objectives, given that we were able to develop a working prototype Framework which demonstrates how ATR algorithms can be optimized. The current prototype may be of limited functionality, but it is not a "mock-up," it is the start of a codebase which we expect to continue building on in Phase II. We conclude that:

- The implemented Framework provides a straight-forward interface for working with ATR algorithms. The Framework leverages the knowledge of the expert ATR algorithm designer, while fully using the power of the computer for optimization purposes.
- The included ATR hierarchy provides structure for implementing algorithms. This hierarchy was developed based on both practical experience and a literature review.

- Knowledge for working with ATR algorithms is embedded into the Framework. This knowledge includes presenting relevant choices for implementations, the handling of input data sets, and other ATR specific issues. As a consequence, the novice ATR algorithm designer will also find the Framework to be a comfortable learning environment.

6.3 Phase II Recommendations

Given the success of the Phase I effort, we recommend a Phase II effort that builds directly on the work done during the Phase I effort. The Phase II work should also develop it further to create an ATR optimization framework that can be used with little or no modification to perform optimization on real ATR algorithms. The objectives of the Phase II effort will be:

- Define functional requirements
- Extend Phase I framework prototype
- Enhance framework with new capabilities
- Acquire an image processing library
- Implement several optimization algorithms, including GA, Powell's method, and the downhill simplex method
- Validate framework by implementing and optimizing two ATR suites
- Implement performance logging and analysis
- Develop a commercialization plan for the technology

Table 6.3-1 below compares proposed Phase II effort with completed Phase I work. It clearly shows how the proposed Phase II effort both builds on and expands the work done in Phase I.

Table 6.3-1: Comparison of the Phase I Effort and the Proposed Phase II Effort

Feature	Phase I	Phase II
Objective	Feasibility analysis	Develop a full-scale working prototype which can be commercialized with minimal additional effort
Focus	Implement a basic framework for ATR algorithm optimization	Enhance and validate a framework for ATR algorithm optimization
Commercialization	Identify potential uses of technology	Work with a suitable machine vision vendor to develop requirements for a marketable prototype Find other markets for product
Application	Implement and demonstrate a simple segmentation task	Implement and optimize two full ATR suites
Resulting product	Prototype Java classes	ATR Optimization Framework
Software platform	Java on Mac	Java on Unix, Windows NT etc. Interface with existing C/C++ code
Use of existing technology	Java GUI features	Java GUI features Third party image processing library
Validation	Validate approach only	Validate using the two implemented and optimized ATR suites
Demonstration	Demonstration of basic GUI and optimization functionality	Demonstration of the full functionality of the Framework
Delivery	Final Report	ATR Optimization Framework Programmers Guide Users Guide & Final Report

6.4 Phase III Commercialization

The Framework for optimizing ATR algorithms can most directly be applied to the optimization of machine vision systems. As a result we are pursuing contacts with machine vision systems suppliers and integrators.

This Framework has potentially broad usefulness within the DoD. The finished Framework could be used to handle the ATR needs of the Navy, Army, and the Air Force. There may also be applications for joint commands such as the Ballistic Missile Defense Organization (BMDO). There might also be a need for some specialized frameworks to match particular applications, such as undersea sonar-based target recognition for the Navy.

The optimization Framework is a general enough idea that it is not limited to ATR or other image based algorithms; we believe that it could be useful in the optimization of any complex system. As a result, we also plan to develop this tool also as a companion to the Company's shipping Learn Sesame personalization toolkit that is currently being marketed to corporate websites.

7 References

ATRWG. (1991). Comparison of Statistical Pattern Recognition, Model-Based and Neural Network Approaches for Automatic Target Recognizer : Automatic Target Recognizer Working Group (27 June 1991).

Ballard, D. H., & Brown, C. M. (1982). Computer Vision: Prentice Hall, Englewood Cliffs, NJ.

Bhanu, B. (1986). "Automatic Target Recognition: State of the Art Survey". IEEE Trans. on Aerospace & Electronic Systems, AES-22(04), 364-379.

Bishop, C. (1995). Pattern Recognition and Neural Networks. Oxford, UK: Oxford University Press.

Bradski, G., & Grossberg, S. (1995). "Fast-Learning VIEWNET Architectures for Recognizing Three-dimensional Objects from Multiple Two-dimensional Views". Neural Networks, 8(7/8), 1053-1080.

Duda, R., & Hart, P. (1973). Pattern Recognition and Scene Analysis. New York, NY: Wiley.

Gonzalez, R. C., & Woods, R. E. (1992). Digital Image Processing. New York, NY: Addison-Wesley.

Grossberg, S., Mingolla, E., & Williamson, J. (1995). "Syntetic Aperture Radar Processing by a Multiple Scale Neural System for Boundary and Surface Representation". Neural Networks, 8(7/8), 1005-1028.

Hecht-Nielsen, R., & Zhou, Y. (1995). "VARTAC: A Foveal Active Vision ATR System". Neural Networks, 8(7/8).

Heitger, F., & von der Heydt, F. (1993). "A Computational Model of Neural Contour Processing: Figure-Ground Segregation and Illusory Contours". IEEE 4th International Conference on Computer Vision, Los Alamitos, CA.

Kasturi, R., & Jain, R. C. (1991). "Introduction to Chapter 2: Segmentation", Computer Vision: Principles, . Los Alamitos, CA: IEEE Computer Society Press.

Koch, M. W., Moya, M. M., Hostetler, L. D., & Fogler, R. J. (1995). "Cueing, Feature Discovery, and One-class Learning for Synthetic Aperture Radar Automatic Target Recognition". Neural Networks, 8(7/8), 1081-1102.

Lindley, C. A. (1991). Practical Image Processing In C: John Wiley & Sons, Inc.

Manjunath, B. S., & Chellappa, R. (1993). "A Unified Approach to Boundary Perception: Edges, Textures, and Illusory Contours". IEEE Trans. on Neural Networks, 4(1), 96-108.

Reisfeld, D., Wolfson, H., & Yeshurun, Y. (1995). "Context-Free Attentional Operators: The Generalized Symmetry Transform". Int. Journal of Computer Vision, 14(2), 119-130.

Ritter, G. X., & Wilson, J. N. (1996). Handbook of Computer Vision Algorithms in Image Algebra. Boca Raton, FL: CRC Press.

Rogers, S., Colombi, J., Martin, C., Gainey, J., Fielding, K., Burns, T., Ruck, D., Kabrisky, M., & Oxley, M. (1995). "Neural Networks for Automatic Target Recognition". Neural Networks, 8(7/8), 1153-1184.

Rosenfeld, A. (1984). "Image Analysis: Problems, Progress and Prospects". In Challappa & Sawchuk (Eds.), Digital Image Processing and Analysis, (Vol. 2,). Los Alamitos, CA: IEEE Computer Society Press.

Schürmann, J. (1996). Pattern Classification: A Unified View of Statistical and Neural Approaches. New York, NY: Wiley.

Sherman, J. W., Spector, D. N., Swonger, C. W., Clark, L. G., Zelnio, E. G., Lahart, M. J., & Jones, T. L. (1993). "Automatic Target Recognition Systems". In Robinson (Ed.), Emerging Systems and Technologies, (Vol. 8,): Environmental Research Institute of Michigan.

Sobel, I. (1970). "Camera Models and Machine Perception". Stanford AI Memo 121, Stanford University, Stanford, CA.

Appendix A: Literature Review of Candidate ATR Systems

We have included in this appendix brief summaries of nine of the many ATR algorithms that were examined in order to develop the taxonomy in section 4. These algorithm are quite varied and represent a broad sampling of ATR algorithms. The summaries include a brief description, some observations, and diagrams that explain how workings of the algorithm. The algorithms included are the following:

ATR 1: "Automatic Target Recognition via the Simulation of Infrared Scenes" by Aaron D. Lantermann, Michael I. Miller, Donald L. Snyder

ATR 2: "Employing Contextual Information in Computer Vision" by Thomas M. Strat

ATR 3: "Vertex Space Analysis For Model-Based Target Recognition" by Azriel Rosenfeld and Gary Whitten

ATR 4: "Model-Based Automatic Target Recognition System for the UGV/RSTA Ladar" by Jacques G. Verly, Dan E. Dudgeon, and Richard T. Lacoss

ATR 5: "Recognition of 3-D objects from multiple 2-D views by a self-organizing neural architecture" by Gary Bradski and Stephen Grossberg

ATR 6: "Multiresolution approaches for automatic target detection" by Bjorn Jawerth and Leonard Mygatt

ATR 7: "Target Recognition Using Multiple Sensors" by Y.T. Zhou and R. Hect-Nielsen

ATR 8: "A Multifeature Decision Space Approach to Radar Target Identification" by J. G. Teti Jr., R. P. Gorman, and W. A. Berger

ATR 9: "Multi-Target Discrimination with Linear Signal Decomposition/Direction of Arrival Based ATR" by Barry K. Hill, David Cyganski, and Richard F. Vaz

ATR 1: "Automatic Target Recognition via the Simulation of Infrared Scenes"

Authors: Aaron D. Lantermann, Michael I. Miller, Donald L. Snyder

Group: Electronic Systems and Signals Research Laboratory, Department of Electrical Engineering Washington University

Field use: airborne detection of land based units

Sensor type: forward mounted FLIR sensors, though the model can be broadened to LADAR as well.

Description:

Unlike many traditional approaches to the ATR problem, the developers of this particular algorithmic suite chose to exploit the paradigm of direct hypothesis matching via the generation of synthetic scenes. Traditionally, ATR algorithms attempt to locate and identify targets through direct analysis of the image scene. Potential targets are first denoted and isolated, then identified and characterized. The novel approach used in this suite, though, determines to identify targets by "imagining" the scene: "pattern recognition = pattern synthesis".

Scenes are engendered from an arrangement of target vehicles via simulated rendering. The rendering incorporates target configurations and emissive characteristics as well as image distortions estimated from sensor statistics. These synthetic scenes are then compared to actual image data by a likelihood function. From the combination of this likelihood along with a prior distribution spanning the set of possible scenes, a posterior distribution arises. The inference engine samples from this posterior to search the sample space for the best matching hypothesis.

Due to the discrete nature of target groupings in terms of number and type and the continuous nature of their positions and formations, the inference engine combines "jumps": quantized state transitions between model subspaces and "diffusions" probabilistic searches of those subspaces, to generate hypotheses. Jumps are defined by three types of discrete transitions: *births* where new targets are introduced, *deaths* where old targets are considered erroneous and expurgated, and *metamorphs* where a target is transformed to a different type. Amidst the jumps, which occur at distributed intervals as determined by a jump intensity function, diffusion processes, corresponding to the discrete target grouping subspaces, samples from the posterior distribution according to a gradient ascent. Together, the two processes constructs hypotheses towards a higher probabilistic match with the actual image data.

Observations:

This is a rather different approach to the ATR problem. However, one thing to note is that there were a rather large set of simplifying assumptions made. Aberrations due to atmospheric effects, occlusions introduced by background and foreground clutter, positional variations arising from terrain formations, etc. have all been removed from initial consideration. Without these very common components of a real battlefield environment, it's premature to say that the experiments conducted demonstrate concept viability. Much of this is due to the fact that the sample space explored by the algorithm are probably orders of magnitude smaller than a realistic sample space. I.e., various non-planar geometric conditions, which infuse an further degree of complexity once vehicles can be oriented off a flat surface, and clutter that can seriously obfuscate target signatures and render matching almost untenable if the entire sample space must be searched to find the "best" hypothesis. However the idea is an interesting one, that has found root in some hypothesis generators for Model-Based classifiers.

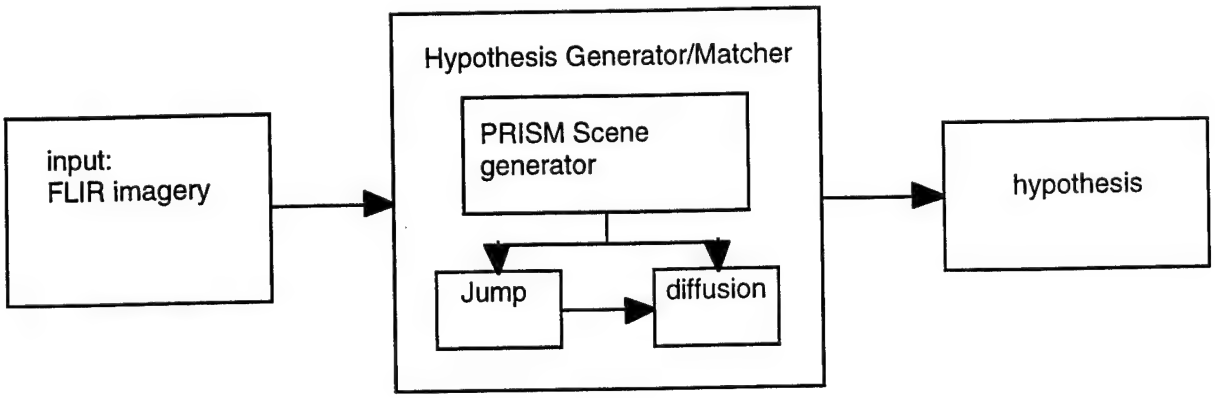

Figure A-1: Schematic of the Hypothesis generating ATR Method

ATR 2: "Employing Contextual Information in Computer Vision"

Author: Thomas M. Strat

Group: Artificial Intelligence Center, SRI International

Field use: One of the systems employed identified objects in nature imagery, analyzing the arboreal scenes to extract and identify trees and other flora or fauna. The primary purpose of the other system was to detect buildings and other man-made structures, not necessarily military in nature or origin.

Sensor type: not specified, perhaps CCD, at least for the nature recognition system

Description:

Contextual information is indispensable to the proper functioning of ATR systems. Any information not derived directly from image data falls within the category of contextual information: mission objectives, weather conditions, intelligence reports, battlefield conditions, etc., more or less anything that defines the image environment. Context provides invaluable insight into the composition and meaning of a scene. Coupled with the raw information extracted from the scene itself, contextual clues can greatly augment the effectiveness and efficiency of an ATR system. Most ATR algorithms incorporate a number of contextual clues as implicit assumptions that determine certain algorithmic parameters, algorithmic sub-blocks constituency, even algorithmic composition and internals. However, in the scope of a general ATR scheme, it seems too limiting to confine inspection to a narrow set of assumed predefined conditions, where the most minute change in scene context could render the system ineffectual. The approach explored in this paper attempts to devise a context ontology and a decision/control architecture that incorporates the explicit contextual information into every step of the ATR process.

Contextual information is divided into three categories: physical, photogrammetric, and computational. Physical context defines the spatial environment: mountainous, forested, even specific details like a boulder sits at (x,y). Sensor attributes and image acquisition conditions comprise the photogrammetric context: field of view, orientation, time and date, etc. Computational context includes resources available, extent of processing, etc. The determination of just what contextual information defines the operative context for a given IU sub-block depends heavily on the application domain. When selected these contextual parameters find their representative incarnations as assertions, both established and hypothetical.

At the heart of the system lies a control model that determines the applicability of IU components given the context, selects the best algorithm for the situation, and constrains its parameters. The general structure of the system builds off the sequential ATR paradigm. From a collection of IU algorithms that perform the current analytic step, the sub-block that best fits the task, world data, and world knowledge/model context is chosen. The resultant scene description resulting from the IU processing is examined with the world knowledge and if consistent, enriches the world knowledge. In this way context determines the ATR pathway with a global scope.

One implementation of this general architecture utilizes context sets, or more generally context tables to define the required/preferred context parameters for each IU algorithm. Context set groupings are used to control three IU steps: candidate generation, candidate comparison, clique formation. The pool of candidates obtained from the first step are sorted into a partial order at the second step. Then a clique is constructed from the "best" candidate from each partial order, testing consistency with the world model/knowledge, rejecting the inconsistent candidates. Finally the "best" clique is selected as the representative model for the scene. At every stage the context sets are used as satisfaction criteria to determine which algorithm is suitable for the current step. Moreover, the information generated at each step becomes a part of the global context, which can then further refine the model hypothesis. All the knowledge accrued as context and hypotheses are stored in a persistent OO-database that serves as the world knowledge/model.

Observations:

Perhaps one thing to note about this system is that when tested against imagery taken from a limited 2-mile square region, recognition was quite good. However when applied to images taken from different nature locales, performance dropped off by large margin. This was largely due to the limited relevance of the context knowledge to the new settings. Here enters the problem of defining context scope. Using a context scope localized to the present application domain restricts the generality of the system. Enlarging the scope may drastically increase the computational cost by giving each IU sub-block a very large context set. A more fundamental question is the very process of divulging contextual information and determining it's relationship to the IU algorithms. The very extensibility that the system is intended to allow may be its limitation. Other observations include the utility of adding in a learning mechanism for creating new context hypothesis and synthesizing new context inferences, incorporating the context information directly in the IU algorithm, not just in the control structure since creating algorithms for a every possible limited context seems atomization gone overboard.

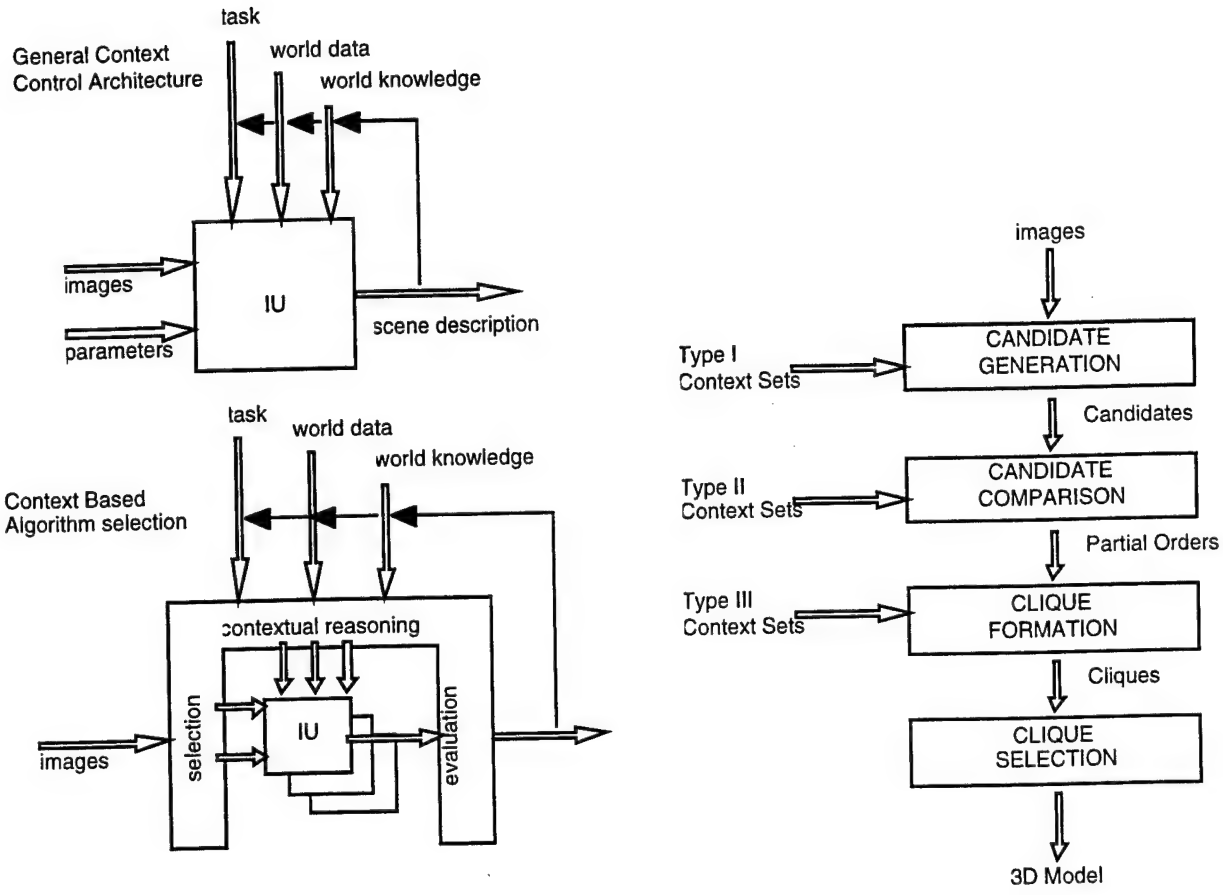

Figure A-2: Employing Contextual Information

ATR 3: "Vertex Space Analysis For Model-Based Target Recognition"

Authors: Azriel Rosenfeld and Gary Whitten

Group: Center for Automation Research, University of Maryland

Field use: not specified, but tested on land-based tower-ground images

Sensor type: visible and infra-red instrumentation

Description:

At the heart of this algorithm is a feature representation called Vertex Space. Model-Based ATR systems can generate multifarious and detailed hypothesis from abstract representations and perform robust verification analysis based on feature matching between the image and model feature space. One problem that often plagues Model-Based recognition is the robust extraction of features from the raw image. Often times the extraction process is either computationally intensive or susceptible to erroneous detection. The Vertex Space feature representation strives to solve this by reducing the 3-D model to a set of vertices (curvature maxima) that remain invariant to 4 out of the 6 degrees of freedom in a 3-D viewing geometry. This alleviates both the computational strain of generating and matching complex 3-D models, and the resource strain of having to internally store the cumbersome models.

Vertices, in the Vertex Space model, are very similar in nature to geometric vertices. A Vertex occurs at the intersection of two straight lines or at the point of maximal curvature of a continuos boundary. Size (degree measure of the angle between the rays), orientation (relative angular location), and location uniquely define each Vertex. While location may be highly variable from image to image, size and orientation are invariant with respect to translation. Although rotation in the image plane shifts the orientation value, it is only by a constant factor since the measure is relative to an arbitrary axis. Thus, only azimuth and elevation angles will change the Vertex Space representation of a given 3-D model. This reduction in model variables allows for a compact representation that expedites correlation matching by shrinking the search space (decreasing complexity), cooling the computational intensity of correlation calculation, and enhancing the resistance to clutter and occlusion (vertex independence allows for strict template matching to ignore clutter features).

Precomputing the Vertex Space representation of targets from their 3-D + sensor features view to 2-D projection, augments matching efficiency by directing the candidate search process via hash table indexing. From a uniformly distributed set of viewpoints residing on an observation sphere

surrounding the 3-D target model, a set of 2-D Vertex Space views are generated and indexed into a hash table by their constituent Vertices (O,S). Candidate hypothesis generation then proceeds as follows: 1) every extracted Vertex from the image is indexed into the table, and the resultant views are incorporated into a view histogram. The peaks in the histogram indicate likely view candidates. 2) indexing into a second hash table using image and model Vertex pairs generates an orientation offset histogram, where the peaks are again labeled as likely offset values. 3) using positional alignment data that is stored with each model Vertex: angles formed with other Vertices, a consistency check is made using a consistency matrix to find mutually consistent vertices. After a candidate hypothesis passes all this checkpoint, the corresponding 3-D model object is instantiated and matched with the image directly, for a final, thorough hypothesis verification. The Vertex Space hypothesis guides the final geometric target orientation by using the spherical coordinates as a least-square approximation to the real geometry. Thus the indexing process using Vertex Space directs the search into 3-D model space, increasing specificity and decreasing the information load at each level, thereby streamlining the process of hypothesis generation, selection, and verification.

Crucial to the success of the Vertex Space Model-based recognition engine is the robust extraction of Vertex features from the image. The extraction process relies on a local curvature maxima calculation to pinpoint potential Vertices. First the raw curvature produces an orientation field that depicts the contour flow of the image. A non-linear low pass filter then removes isolated points of high curvature that arise from noise. Next, conditional gradient processing of curvature and orientation identifies boundaries for region splitting. Gradient thresholding partitions the curvature regions by the identified boundaries. Curvature regions without adequate spatial support are rejected through a size filter. Finally, a vertex location is chosen for each remaining curvature region by determining the most deeply embedded pixel.

Observations:

The system described by this algorithm only specifies a feature extractor and classifier, leaving the preprocessing and segmenter to be determined. The intent was to investigate a possible means of achieving the elusive goal of real-time battlefield ATR, where adverse conditions, high clutter, and sparse data availability prevail. It seems that system is successful in augmenting resistance to artifacts generated by occlusion and clutter, due to Vertex independence.

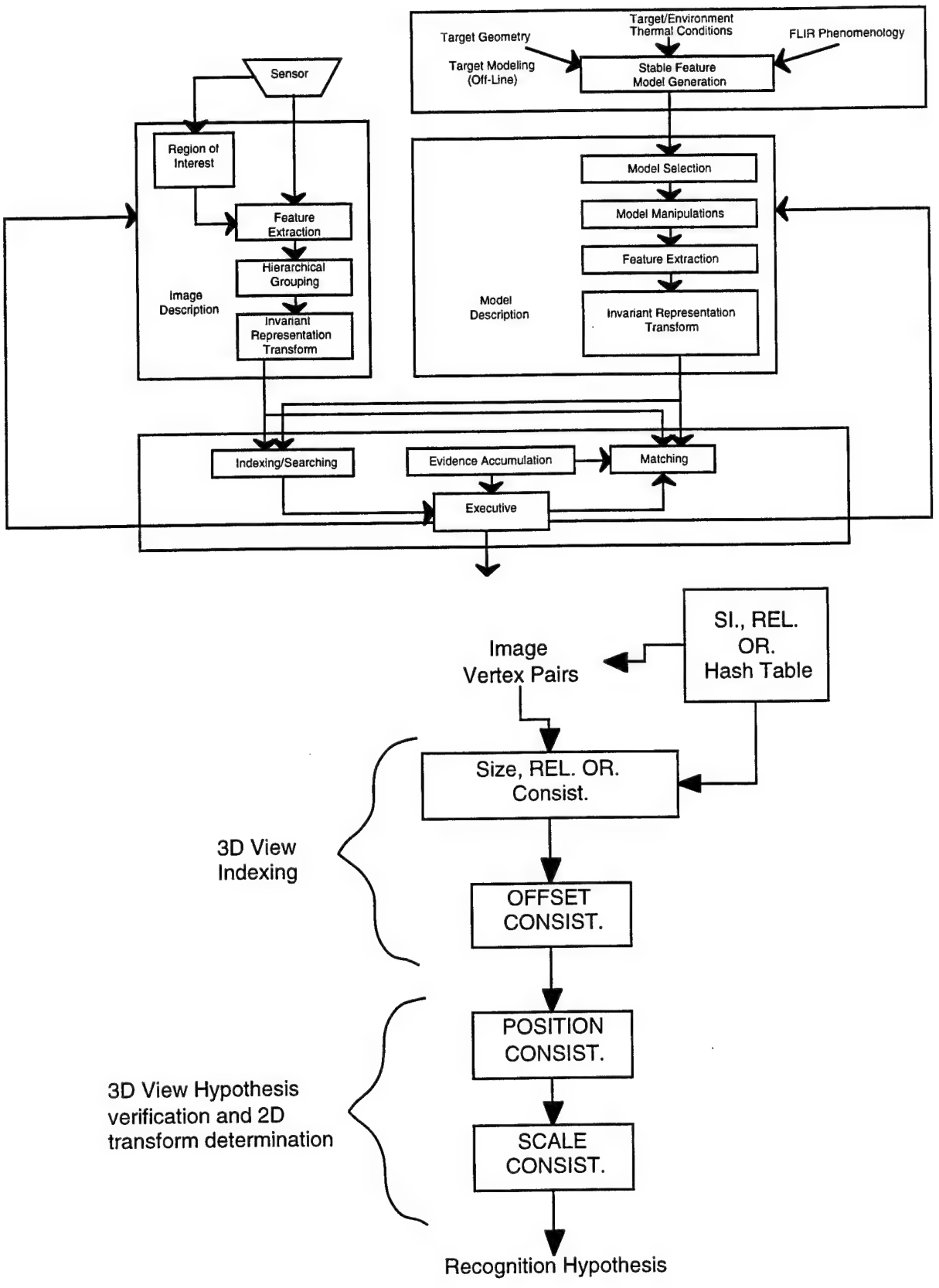

Figure A-3: Model Based ATR

ATR 4: "Model-Based Automatic Target Recognition System for the UGV/RSTA Ladar"

Authors: Jacques G. Verly, Dan E. Dudgeon, and Richard T. Lacoss

Group: Machine Intelligence Technology Group, MIT Lincoln Laboratory

Field use: for real-time use on a Semiautonomous Surrogate Vehicle direct navigation

Sensor type: LADAR, forward looking

Description:

This system relies on Functional Template Correlation as the heart of its recognition process. Functional Templates are image correlation functions defined for a target template. For each pixel value in the template, a function $f_n(x)$, that maps an Image pixel to a value in a given range [0,255]. These values are then averaged across the template and clipped. The template is applied to every pixel in the image and can also be rotated to search for rotated patterns. Due to the nature of the Functional Templates used, the segmented regions must be reprojected to a top down view. One caveat in this approach is the loss of target information as the view is reprojected, as there is self-obscuration i.e. a frontal view of a tank that becomes a mere set of lines delineating the forward perimeter only. These situations are dealt with by ...

Target detection examines the image for Interest Images, regions that are likely to contain targets. These interest images, drawn from various image modalities using heuristic directives, are then combined via some fusion scheme to obtain a complete collection of Interest regions, which traverse the gamut of image modalities including images that weren't even used to generate the Interest Images. The primary heuristic detection schemes used is height limited-verticality: that finding objects jutting out vertically from some reference plane, limited by an upper height bound to confine interest to the range of military vehicle sizes. Another use of the Interest Images comes from determining a range based segmentation where pixels of similar range and high interest divide the image into range-segment. Together the two combine to form an RS-window which bounds regions of interest in a given range segment. Extraction of the target silhouette from RS-window is accomplished by cutting out the portion of the image in the RS-window corresponding to the range-segment associated with the window.

Recognition is preceded by vertical reprojection to transform the imagery into a form tailored for FTC. The overhead view resulting from this procedure is accompanied by a trace image, which is a

mask of the reprojected pixels, a count image, which counts the number of pixels reprojected to a given pixel, and a z-coordinate image, depicting height relative to some chosen ground plane. With the reprojected images, recognition proceeds by first applying a Hough transform to the count image to determine target orientation. This shrinks the angular range that FT's have to be spun at each pixel point since the general orientation of the target is known. Then the FTC kicks in. For each type of overhead view to be analyzed, a different set of FT's are needed: count FT's, z-coordinate FT's , etc. Within each set every FT corresponds to a azimuth angle range. The FTC engine then searches through the image to find the highest correlating FT, and thereby labeling the target.

Observations:

One interesting thing to note about this particular system is that due to the unavailability of the actual working sensor for the UGV, many sensor specific portions of the algorithm were skipped over, like sensor generated noise, imaging characteristics, etc. The inherent dependence upon reprojection to obtain some sort of invariant imaging for FTC seems somewhat debilitating, especially when the available views are normal to the face of the target which translates to a severely deprecated reprojected image.

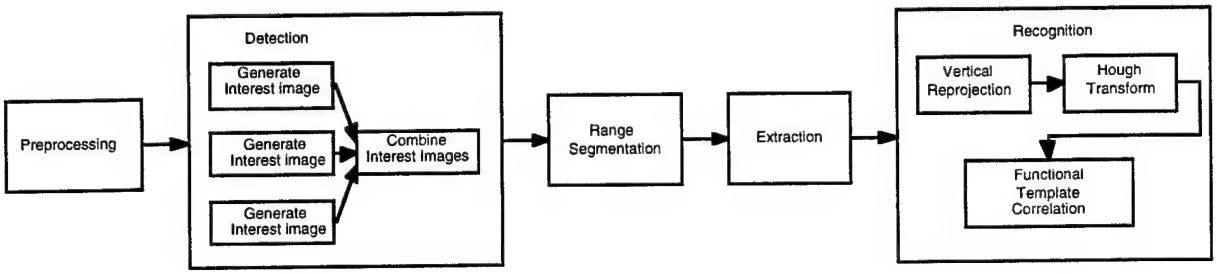

Figure A-4: Model Based ATR Using LADAR

ATR 5: "Recognition of 3-D objects from multiple 2-D views by a self-organizing neural architecture"

Authors: Gary Bradski and Stephen Grossberg

Group: Boston University Center for Adaptive Systems and Department of Cognitive and Neural Systems

Field use: intended as a general system for ATR, unspecified as to particular application. Testing performed using 2-D jet aircraft imagery

Sensor type: unspecified

Description:

From input to output, the core of this system is based on a neural network architecture called VIEWNET. Everything from object detection all the way through object classification is processed using some neural network system. Initial processing occurs using a CORT-X 2 filter. Filtering removes image noise and traces target boundaries, resulting in a refined, isolated target image. A log-polar transform is then applied to the segmented image with respect to the object centroid, and re-centered to generate a 2-D scale and rotation invariant representation. The invariant images are then coarse coded using Gaussian filters to eliminate remaining noise, reduce foreshortening effects, and increase generalization. At the final step, these compressed images are sent through a fuzzy ARTMAP learning engine.

The CORT-X 2 filter is divided into 2 steps: illuminant noise filtering, and boundary segmentation. An on-center and off-center filtering network combine to enhance image contrast and eliminate variable illumination. The two filters have polar affinities for concave and convex corners in the image, which facilitates the following boundary segmentation process. Large scale and small scale transforms of the convolved images are applied to confer additional noise reduction and positional localization sensitivity. The two sets of images, large and small, are then passed through a series of hypercomplex cells for edge detection in parallel. At the end of the pipeline the two resultant boundary definitions are combined to produce the extracted target boundary.

Dividing the 2-D boundary image by its 1st and 0th moments then subtracting off the center ascertains the figure centroid, which is then used to shift the image to its center. Applying the log-polar transform to the image with respect to the center yields coordinates of log radial magnitude and angle: $re^{i\theta}$. Figure scale and rotation become constant shifts in this representation. Using these shifts to center the transformed image imbues the representation positional, angular, and

dimensional invariance.

The next step of coarse coding serves to compress the image data, compensate for alignment inaccuracies, reduce foreshortening and self-occlusion effects, and expunge noise. However there is the potential of over simplifying the image to the point of losing image features crucial to the recognition analysis. Treading upon this fine line is directed by a spatial averaging method that convolves the image with a Gaussian function φ, then sampling the resultant image with delta functions every T pixels. Compression can shrink the image down to as small as 4x4 pixels.

The fuzzy ARTMAP applied to the coarse-coded image to classify the target utilizes an ART module to categorize the input vector, which is then mapped by an associative memory to output nodes predicting the target type. During supervised learning, the network receives an input containing both the image vector and the truth classification data. If the image is incorrectly classified, a memory search is invoked by a mechanism called match tracking, where either a pre-existing category mapping to the correct target type is found or a new category is created and further learning occurs to form the connection. At the output end, the maximally activated node indicates the predicted target type.

Observations:

One key assumption/simplification made by this system is that the target is separated from its background. This is most definitely cheating, because the high contrast then allows for near-perfect object detection and segmentation, irrespective of the method used, as long as its viable. False alarms and missed detections are completely removed from the picture, and an inaccurate view of system performance is imposed. Target detection is a crucial part to most ATR's; making this assumption basically removes one of the inherent difficulties of the field from consideration, since many real battlefield images will not have such high contrast, especially when targets are camouflaged or atmospheric conditions occlude the target signature.

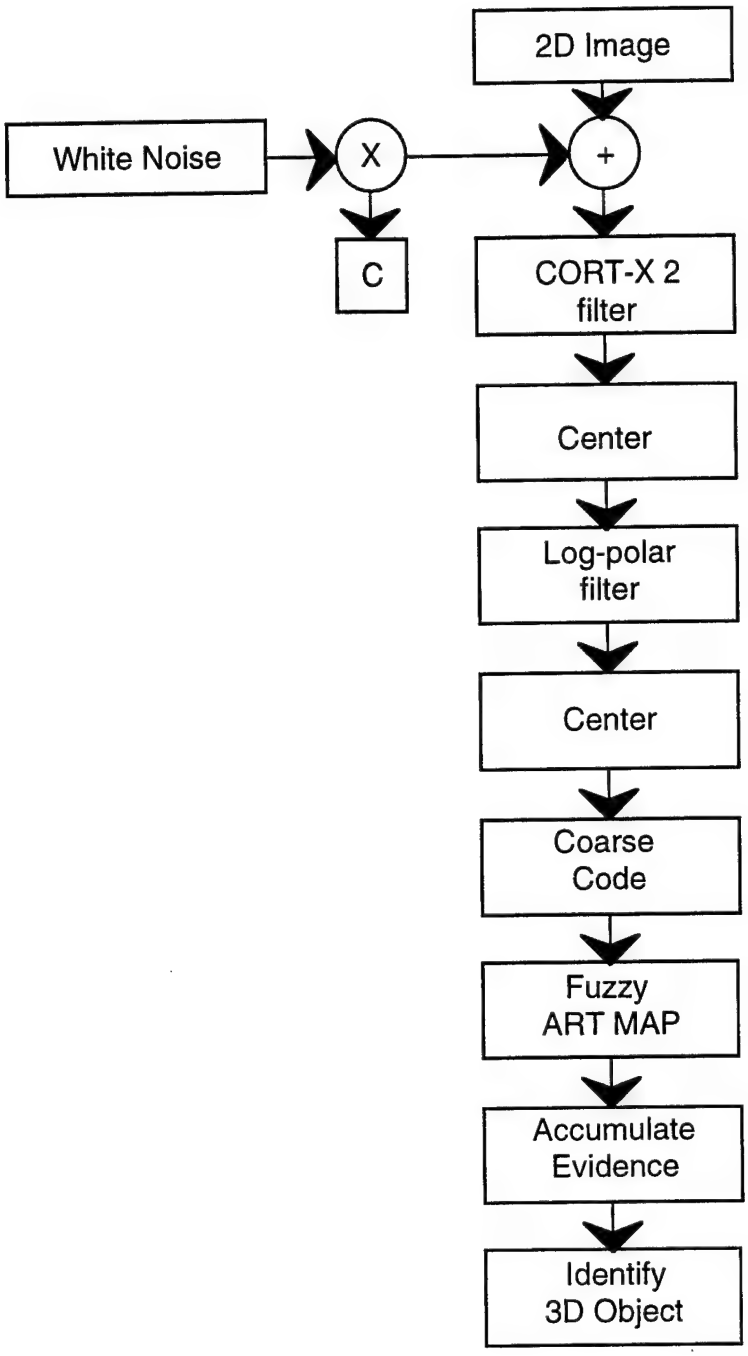

Figure A-5A: Overall Architecture

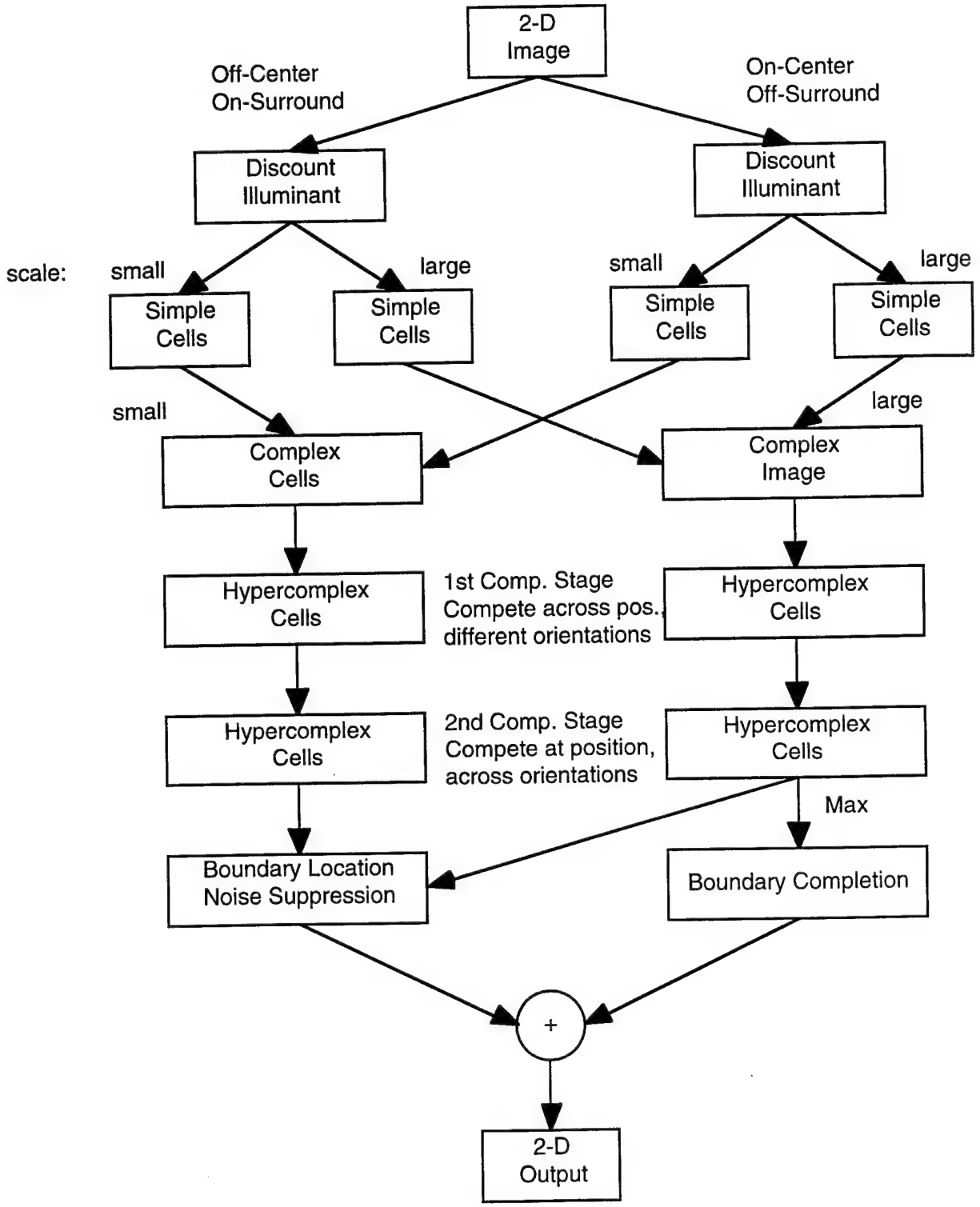

Figure A-5B: CORT-X 2 Filter

ATR 6: "Multiresolution approaches for automatic target detection"

Authors: Bjorn Jawerth and Leonard Mygatt

Group: Summus Ltd.

Field use: not specified

Sensor type: not specified

Description:

As of the report, the system had only a target detection/segmentation scheme and feature enhancement algorithms. Using a wavelet-based oscillation measure computed as a weighted sum of the wavelet coefficients and DC component coefficients over a varying neighborhood, image texture properties can be characterized and defined. From this measure, the "fractal signature" of the underlying texture is used to identify the clutter as an n-dimensional surface. This surface can then be used as a decision threshold or feature for subsequent detection and classification algorithms. The segmentation algorithm that follows partitions the input image according to homogenous regions based on pixel intensity.

Two approaches are used in feature enhancement and extraction: wavelet domain transforms and applications of linear/non-linear PDE's. In the first method, conventional high-pass filters applied to small pixel neighborhoods in the wavelet domain augment and identify textures. The second method uses a PDE defined as $u_t = -|\nabla u| F(L(u))$ where L is an edge detection or curvature based operator, and F is a step function. Solving this equation produces an image with sharper edges and less noise. Essentially the original image is treated as a blurred facsimile. Reversing the diffusion process restores the detail inherent to the scene. Enhancement quality depends on the choice of the L operator.

Observations:

Since this was only a progress report, not too much can be said about the system as a whole. The testing was limited, but indicated very good performance in terms of noise reduction and enhancement.

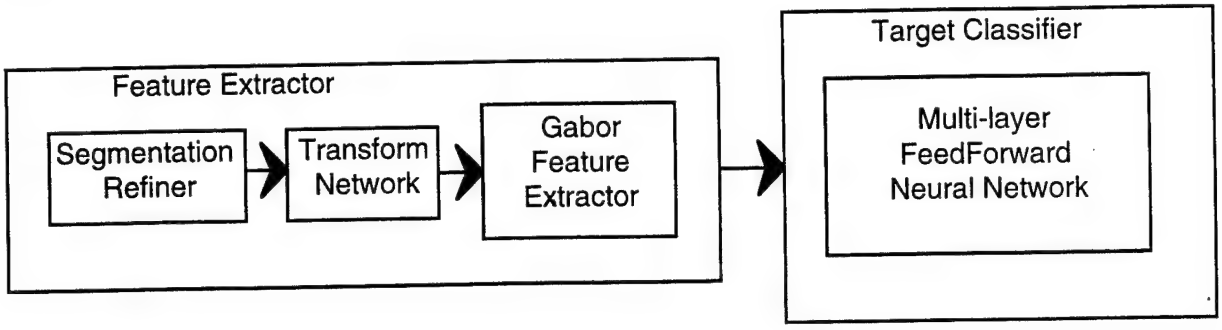

Figure A-6: Multiresolution ATR

ATR 7: "Target Recognition Using Multiple Sensors"

Authors: Y.T. Zhou and R. Hect-Nielsen

Group: HNC, Inc.

Field use: not specified

Sensor type: applicable to various sensor modalities

Description:

This system includes a feature extractor and classifier, comprising a target recognition sub-system that can be incorporated in an end-to-end ATR system. The feature extractor focuses on geometric-variant features, based on the noise resistance qualities of such attributes. To reduce the computational cost of the classification process that raw geometric properties often incur, a spatial domain transform is applied to the feature space data. Classification is accomplished via a multi-layer neural network architecture capable of synthesizing input data from multiple sensors into a candidate identification hypothesis.

There are three major components to the feature extractor: segmentation refiner, transform network, and Gabor feature extractor. The segmented input often comes somewhat mottled, with inexact target boundaries, holes, etc. Refining smoothes the boundaries and makes them more regular, framing the target with a much tighter fitting border. Holes are refined by the use of morphological operators that preserves the continuos nature of target image signatures. The transformation relies on a what-and-where inspired neural network that takes an object image, rotates it along the optical axis, scales and translates it to a centered canonical position. What remains is an image varying only with azimuth and elevation. Estimates of the target position taken from the segmented image using pyramid filters are used to reposition the original image. This transform operation is highly resistant to noise. Gabor features are computed using Gabor functions, Gaussian weighted sinusoidal functions. Due to the parametrized nature of the functions, which allow for orientation along different directional axes, center points, and spatial ranges, multiple features can be computed from varying perspectives. The sine and cosine components correspond to the presence of bars and edges. Computation at several points in and surrounding the center gives a rich set of features useful for classification.

Classification is accomplished by a multi-layer feedforward neural network. The network is typical of most neural architectures. Input and output layer sizes are predetermined by feature set and target class set cardinalities. Hidden layer depth and size though must be determined

experimentally, along with the weights and other parameters. Training adjusts these values according to a back-propagation learning algorithm.

Observations:

For the most part, this a fairly straight forward neural network approach to ATR.

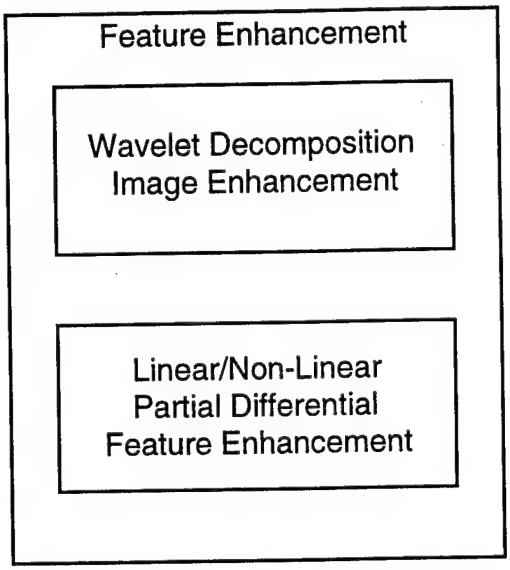

Figure A-7: ATR with Multiple Sensors

ATR 8: "A Multifeature Decision Space Approach to Radar Target Identification"

Authors: J. G. Teti Jr., R. P. Gorman, and W. A. Berger

Group: JJM Systems, Inc. and Dept. of Physics and Electronics Engineering

Field use: airborne target identification

Sensor type: sensor suite spanning a spectral range suitable for characterizing the target; single monostatic radar sensor opening at X-Band

Description:

This ATR system uses a classical formulation to the target identification problem. From the sensory input acquired from the sensor suite, a set of target features are extracted according to target phenomenology. This multidimensional vector of features is then mapped into a decision space constructed from a probabilistic analysis of the feature space and it's correspondence to target identities, giving a final target hypothesis.

The target features used in this particular implementation are radar cross section, length, two spectral features extracted from jet engine modulation, target velocity, and altitude. Together, these features form a 6-dimensional target description vector, which should uniquely identify the target in the mapping. However, this is very much dependent on the specificity of the features themselves in characterizing the targets, contingent upon the very nature of the targets themselves.

Construction of the decision space relies on the ability of the features to partition the state space into N disjoint, distinct regions with sharp boundaries. The regions defined by this partitioning are distributed throughout the state space according to the joint intersection of individual target outcomes from each discriminant feature. This is achieved using a statistical description of the joint probability density function for each target, assuming feature independence. With the target regions delineated in the decision space, application of a Bayesian Decision Rule yields a hypothesis corresponding to the target with the maximal conditional probability.

Observations:

Due to the fact that this system deals with airborne target identification, some of the assumptions made are inapplicable to the ground-based ATR domain. Within the scope of the paper discussion there was no mention of target detection and segmentation, which is a product of the simplified segmentation process in aerial scenarios. Another note is the potential weighting of features in the decision space, to emphasize the more salient and discriminatory attributes. This

would require a separate Bayesian Decision Rule for each decision subspace. Terminal combination of the individual hypothesis according to a weighted scheme would be necessary for ascertaining the final target hypothesis. Lastly, correct modeling and selection of features is crucial to the accuracy of the detection system.

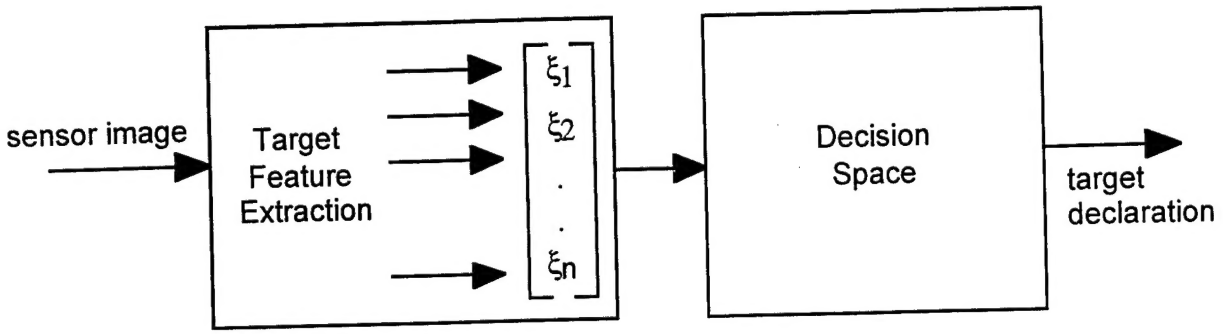

Figure A-8: Multifeature Decision Space Approach to Radar ATR

ATR 9: "Multi-Target Discrimination with Linear Signal Decomposition/Direction of Arrival Based ATR"

Authors: Barry K. Hill, David Cyganski, and Richard F. Vaz

Group: Machine Vision Laboratory, Electrical and Computer Engineering Department, Worcester Polytechnic Institute

Field use: ground based detection amidst high clutter

Sensor type: SAR

Description:

Model based ATR systems exploit geometric or spatial features to generate target hypotheses. However most approaches are either too costly or too difficult to characterize correctly. One slant on the problem is to amass an enormous target image database that spans the entire range of possible target views and orientations. Scanning the database using a correlation match operator on the input image and the database image could require immense computational resources, as the search space blossoms exponentially with the inclusion of additional targets and variables. Another popular approach is to extract pose invariant features, or constrain geometric variance to a smaller, more tractable number of variables, which greatly diminishes storage and computational overhead. An inherent problem, though, is the relevance of the chosen features as target discriminants, since there is a degree of information loss in the transformation process. In this particular system, a novel approach to the model-based paradigm is examined that does not rely upon this search/match approach. Rather, it uses a Linear Signal Decomposition/Direction of Arrival algorithm for engendering both target hypotheses and pose estimates. This method exploits the variance induced by pose to produce these estimates, calculating the target signature as a function of the pose.

Much of the efficiency and power found in the LSD/DOA method lies in its precomputation of Reciprocal Basis Sets. Reducing the target data into a small collection of RBS's allows for an entire target, including orientation variability, to be characterized by a functional model. This functional model is an inner product of the discrete fourier transform of the target view sequence, which incorporates orientation information into the model itself. When combined with the target image, the RBS generates a statistical metric sufficient for matching criterion and a pose estimate, called Synthetic Waveform Samples. These SWS's are so-called due to their interpretation as plane waves that characterize object pose when analyzed with a Direction of Arrival algorithm to determine the directional cosine and it's period. Once these RBS are computed from the basis set of target images along with image and sensor noise estimates, using a matrix Singular Value Decomposition method,

they can be used as image operators, on the input data to determine the pose/identity hypothesis.

To perform the final target recognition step, a Generalized Likelihood Ration Test is used to provide the basis for a decision rule. This Bayesian formulation computes a ratio between the various conditional hypotheses of target identity and the resultant SWS vector. The SWS --> target match with the greatest correlation score wins.

Observations:

Given the results of the study, it appears as though this approach has a very high accuracy in prediction rating, generating a highly level of true + predictions. However it can be noted that the imagery used was not of real battlefield scenes, but rather of SAR images of target vehicles overlaid on man-made cluttered backgrounds. Moreover the entire target detection and segmentation problem was overlooked, because of the focus on the target identification problem. Another point to be made is the small set of targets used, and the singular use of targets in the test imagery, with only one target in each image, I believe. Thus these results may not be completely representative of the system performance.

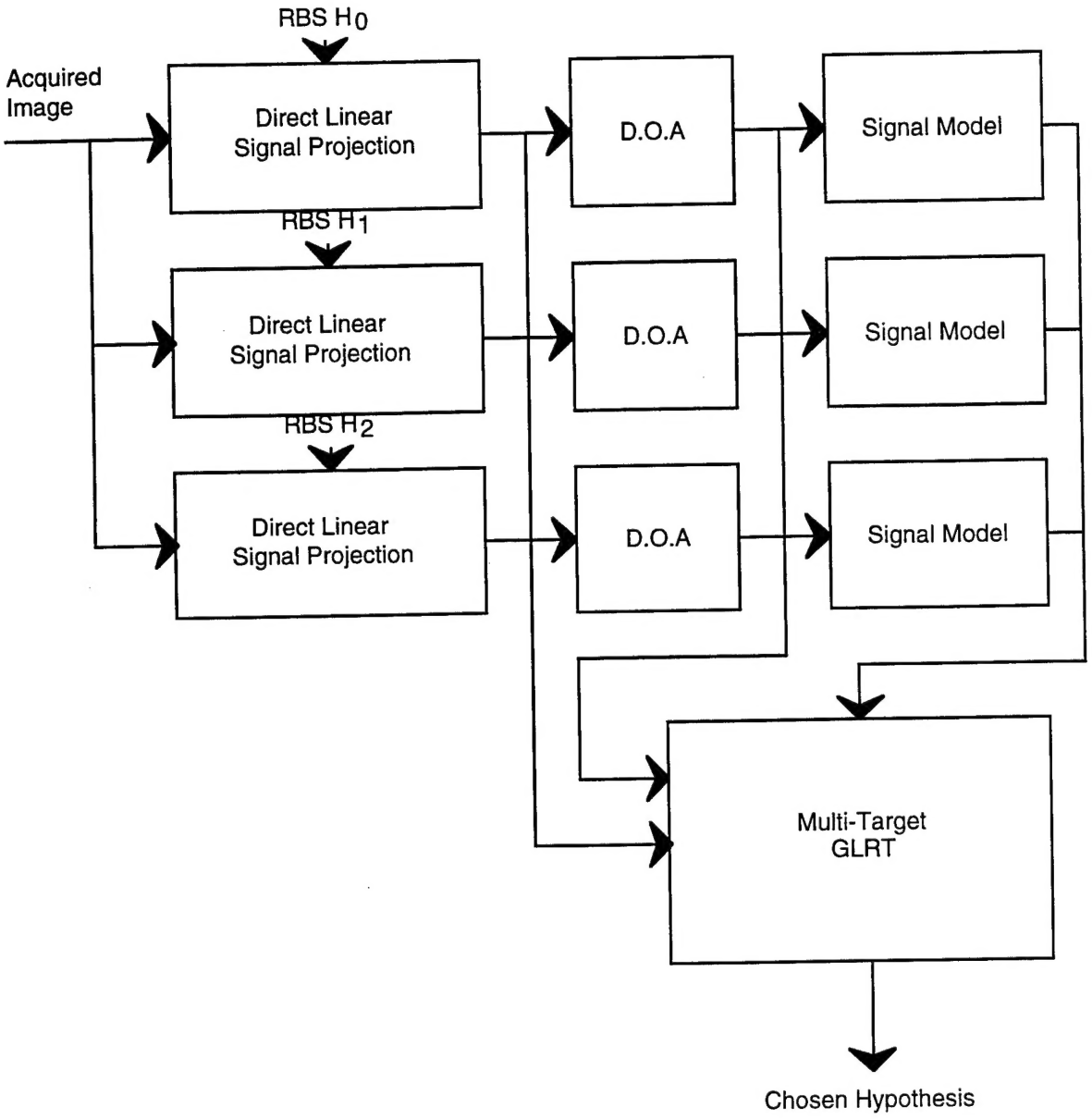

Figure A-9: Multi-target ATR Using Linear Signal Decomposition